December 1981

Simultaneous Equations in the Model System With an Application to Econometric Modelling

Richard Gary Greenberg
University of Pennsylvania

Simultaneous Equations in the Model System With an Application to Econometric Modelling

Abstract

This report describes modifications to the MODEL language and processor to facilitate automatic implementation of solution procedures for systems of simultaneous equations. MODEL is a very high level nonprocedural language for specifying computational tasks. The MODEL processor compiles a specification in the MODEL language into a computer program in PL/I . The purpose of the current modifications is to allow users with relatively little programming expertise to solve complex mathematical systems involving sets of simultaneous equations quickly and efficiently using an automatic program generation approach to modelling.

The primary application which has motivated these modifications is that of Project LINK, an international econometric model composed of independent constituent country/region models which are linked together into a complex network of simultaneous equations.

This report presents the rationale behind these modifications, describes the syntax, semantics, scheduling and code generation of specifications containing simultaneous equations, and illustrates these facilities in applications to two small national models from the LINK system and to a novel linkage mechanism used to simulate trade among the national models.

Comments

University of Pennsylvania Department of Computer and Information Science Technical Report.

Report Number unknown.

UNIVERSITY of PENNSYLVANIA

PHILADELPHIA 19104

The Moore School of Electrical Engineering D2
DEPARTMENT OF COMPUTER AND INFORMATION SCIENCE

TECHNICAL REPORT

SIMULTANEOUS EQUATIONS IN THE MODEL SYSTEM WITH AN APPLICATION TO ECONOMETRIC MODELLING

by

Richard Gary Greenberg

Prepared with Support from
The National Science Foundation
under Grant MCS-79-0298

Joint Computer Science-Economics Project

UNIVERSITY OF PENNSYLVANIA
THE MOORE SCHOOL OF ELECTRICAL ENGINEERING
SCHOOL OF ENGINEERING AND APPLIED SCIENCE

SIMULTANEOUS EQUATIONS IN THE MODEL SYSTEM WITH AN APPLICATION TO ECONOMETRIC MODELLING

RICHARD GARY GREENBERG

Philadelphia, Pennsylvania

December, 1981

A thesis presented to the Faculty of Engineering and Applied Science of the University of Pennsylvania in partial fulfillment of the requirements for the degree of Master of Science in Engineering for graduate work in Computer and Information Science.

Noah S. Prywes, Thesis Advisor

Aravind K. Joshi, Chairman, Graduate Group,
Department of Computer and Information Science

ABSTRACT

This report describes modifications to the MODEL language and processor to facilitate automatic implementation of solution procedures for systems of simultaneous equations. MODEL is a very high level nonprocedural language for specifying computational tasks. The MODEL processor compiles a specification in the MODEL language into a computer program in PL/I. The purpose of the current modifications is to allow users with relatively little programming expertise to solve complex mathematical systems involving sets of simultaneous equations quickly and efficiently using an automatic program generation approach to modelling.

The primary application which has motivated these modifications is that of Project LINK, an international econometric model composed of independent constituent country/region models which are linked together into a complex network of simultaneous equations.

This report presents the rationale behind these modifications, describes the syntax, semantics, scheduling and code generation of specifications containing simultaneous equations, and illustrates these facilities in applications to two small national models from the LINK system and to a novel linkage mechanism used to simulate trade among the national models.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 THE MODEL SYSTEM

This report describes modifications to the MODEL language and processor to facilitate automatic implementation of solution procedures for systems of simultaneous equations. MODEL is a very high level nonprocedural language for specifying computational tasks. The MODEL processor compiles a specification in the MODEL language into a computer program in PL/I. The purpose of the current modifications is to allow users with relatively little programming expertise to solve complex mathematical systems involving sets of simultaneous equations quickly and efficiently using an automatic program generation approach to modelling.

The primary application which has motivated these modifications is that of Project LINK, an international econometric model composed of independent constituent country/region models which are linked together into a complex network of simultaneous equations. The design

described in this report is intended to enable economists working on LINK and scientists working in other areas of expertise to specify such models using the language of mathematics, and to obviate as much as possible detailed knowledge of numerical methods, programming and computer science.

Presently, professionals in social, natural and engineering sciences must acquire significant expertise in statistics and numerical methods as well as in computer systems and programming in order to use a computer to perform complex computations. These additional requirements tend to retard research and development in the users' primary fields of interest. Such requirements can be greatly relaxed by building this knowledge into an intelligent computation system. The MODEL system has built-in intelligence in two broad areas:

1) Mathematics -- consisting of checking completeness, non-ambiguity and consistency of specified data and mathematical relationships, and automatic incorporation of implied statistical and numerical methods.

2) Computer Programming -- consisting of automatic generation of correct and efficient computer programs which implement the mathematically described computations.

The MODEL system's use of a very high level language for specifying data types and computational relationships provides a way to represent models easily and economically

in a language which is quite natural for researchers in the sciences and engineering. A specification of a computation consists of a set of statements describing the schemata of data employed and the mathematical relationships existing among the data. The statements need not be ordered in a way which represents specific steps in the computation, but rather can be presented in any order which seems most natural to the modeler. In specifying equations, the user can employ the syntax and semantics of regular, matrix and boolean algebras. The language provides great flexibility in the definition of data structures, types and names, and in the use of the general language of mathematics to describe equational relationships among these data structures. This is a great advantage over some widely used modelling systems, especially in the area of econometric modelling, which force the user to comply with fixed and sometimes unnatural data base structures and equation formats.

Also in contrast with most of today's econometric modelling tools, the MODEL system uses a compilation approach to automatic program generation rather than an interpretive execution of user equations. This allows us to perform much more complete and in-depth analyses of users' specifications in order to discover as early as possible many different types of mathematical ambiguities, incompletenesses and inconsistencies. Furthermore, the MODEL processor can recognize the need for numerical and statistical solution techniques implied by the user, and

incorporates such techniques automatically into the solution program. Compilation also enables us to produce several types of output reports which completely document the user's specification and which are very helpful in understanding the underlying mathematical model. In addition, the solution program produced during compilation can be reused indefinitely without the penalty of repeated analysis of the specification.

The main product of the MODEL processor is a complete and correct PL/I program which efficiently implements the modeler's requested computations. Our compilation approach further enables us to optimize the generated PL/I program with respect to memory and processing time requirements. When the produced program is compiled by one of today's good optimizing PL/I compilers, the resultant executable program will be extremely efficient. Besides the "normal" optimization, the solution program will have been optimized on a much more global scale by the MODEL system. When large and complex models are being solved, as in econometrics where international models often involve huge statistically derived data bases and tens of thousands of equations, this increase in efficiency can be extremely significant.

1.2 MODEL AND PROJECT LINK

There is a growing tendency in the social, natural and engineering sciences toward increasingly large and complex models. The LINK World Trade Model, the first user of the

currently described system, is typical in this respect. Consisting of local country/region models from twenty-three participating institutions across the globe, the LINK model managed at the University of Pennsylvania is one of the largest and most complete international econometric models. The large scale of the computations is illustrated by the tremendous bank of data on thirty-one national and regional economies and the approximately twenty thousand equations in the system.

1.2.1 SIMULTANEOUS EQUATIONS IN ECONOMETRICS

Iterative solution methods are used to solve the trade models for desired years or periods. The simplest and perhaps most straightforward configuration might seem to group all the model equations into a single iterative loop. In a very large system such as LINK, however, such a large grouping leads to rather inefficient computations. Even in today's atmosphere of shrinking computer time costs, this method would be prohibitively costly. Instead, the equations remain grouped into roughly national divisions, where individual groups of equations exhibit a much tighter coupling that tends to speed convergence of the model. These local models are solved locally iteratively and their solutions are communicated to the linkage model, which performs its calculations of world trade and distributes the resultant data back to the local models for recomputation. This process is continued until desired convergence criteria are satisfied and the system is solved. Such a method leads

to greater computational efficiency as well as greater ease of understanding owing to increased modularity.

1.2.2 PROJECT LINK DEVELOPMENT PROCEDURE

The present procedure used in Project LINK to form a world wide model is to allow independent local development of national models and to perform the combination of the models into a single world trade model and the simulation computations and global analysis at a central location here at the University of Pennsylvania. The Project LINK development procedure can be summarized in the following steps:

Step 1) - independent development and testing of local region or country models by the geographically and politically dispersed organizations which participate in Project LINK. Local developers typically represent variables in their computer models with mnemonic names that are meaningful and easy to comprehend. Local testing is accomplished using exogenous trade variables whose values are supplied by data files.

Step 2) - translation of the local models into a standardized format FORTRAN program suitable for inclusion into the LINK software system. This translation involves changing the organization of data and names of the variables to conform with the LINK software standards. This includes defining all variables in global matrices and referencing the variables with appropriate subscript values, thereby

eliminating any ambiguities which would have resulted from use of the same names for variables in different countries' economies.

Step 3) - Shipment of models to the University of Pennsylvania. FORTRAN programs are compiled.

Step 4) - Project LINK testing and verification of local models. LINK economists attempt to reproduce results obtained by the developers of the models to help ensure the integrity of the software base.

Step 5) - Linkage of local programs with the LINK software system. This generates a computer load module of the world wide system, ready for international simulation.

Step 6) - Test and final computation of world wide solutions. Integration testing and verification and dynamic simulations are performed at LINK central.

The major difficulties in this procedure derive from steps two and four, which involve a great deal of manual labor and are a source of errors which must be found and corrected.

1.3 REQUIREMENTS FOR AUTOMATED TOOLS IN ECONOMETRICS

Experience with the LINK system as it exists and studies of other econometric modelling software systems have engendered a host of desired improvements and extra features which would render such systems more effective and

accessible tools for economists and modelers in other fields of the social and natural sciences and engineering. These areas where improvements are desired include the following:

Use of Natural Variable Names - A software system which enables the user to attach natural, meaningful mnemonic names to the variables he uses would lead to enhanced readability of model software. This in turn leads to better understandability and maintainability as more economists come into contact with the actual model in a purer form than a FORTRAN program with huge nondescript matrices containing the variables.

Ease of Nesting Local Models - An easy, natural way to indicate nesting of local solutions leads to enhanced modularity and greater efficiency of computation. No great expertise in programming should be required to generate or to alter patterns of nesting in the simulation computations.

Choice of Convergence Criteria - Depending on the application, different types of convergence criteria may be applied to iterative solution procedures. Relative convergence criteria, numbers which indicate the maximum allowed difference between successive iteration values of block variables, should be easy to specify. In cases where maximum numbers of allowable iterations are specified as part of convergence criteria, these too should be straightforward to indicate.

Locality of Models - It should be simple and natural to

specify models for local development and experimentation. Economists familiar with local conditions should be able to specify and implement a model with minimum intervention by highly skilled programmers or computer systems analysts. Local data bases, in arbitrarily complex format, should be easy to specify and manipulate.

Modularity - Integration of local models into a linked world trade model should entail a minimum of modifications to the local models. Ideally, the local models should simply be concatenated together with a linkage model without any modifications whatsoever. This improves the speed with which locally updated models may be merged with their trading partners' models and thus helps to increase the currentness of the world trade model. Errors resulting from manual translation of models to other formats are reduced or eliminated by this reduction in required modifications for linking.

Reduction of Required Expertise in Computer Science - Implied in all the above areas, and the most important factor of all is a reduction in the overall level of expertise in numerical analysis, programming and computer science required to generate, test and update econometric models. Economists should be able to directly perform these tasks without intervention by computer science personnel, whose presence in the system is a source of error at worst and an inconvenience at best. Greater involvement of economists in the overall process and an increase of model

"turnover," as time-consuming steps involving interaction with computer scientists are bypassed, could lead to greater freedom to experiment with novel econometric methods and an overall improvement in the timeliness of simulation results.

Generality - Finally, changes should be introduced to expand the generality of modelling systems. Restrictions on the use of new modelling techniques brought about by the standarized model formats required with present modelling software systems impede the progress of research in econometrics and other fields where mathematical models must be forced to conform with packaged software. Ways should be found to implement models in a form more closely resembling the mathematical language in which they are created, rather than in computer languages such as FORTRAN, which is notorious for its illegibility. Such changes would clearly ease the task of modelers and invite participation in modelling in novel ways by heretofore discouraged scientists.

1.4 MODIFICATIONS TO THE MODEL SYSTEM

In response to these and other needs we have sought to apply the tools of nonprocedural specification of computations embodied in the MODEL automatic program generation system to the field of econometrics. The current research has aimed to more clearly understand the needs of modern econometric researchers and to incorporate modifications into the MODEL system designed to meet these

needs. Favorable experience in applying such techniques to other diverse areas has encouraged us to expand the capabilities of the MODEL system into new fields where an unmet need for computer science expertise has greatly impeded the progress of research. In cooperation with the scientists at the University of Pennsylvania's Project LINK World Trade Model Headquarters, we embarked on a series of development stages which can be summarized in the following six steps:

1) Generalization of the MODEL language to facilitate specification of computation of solutions of sets of simultaneous equations. The new syntax and semantics should allow specification of nested iterative solution procedures as outlined above and should allow a modeler to conveniently select from among several iterative solution methods for his computations. Specification of such factors as convergence criteria, maximum number of allowed iterations and initial values for variables in iterative solution procedures should be possible in a natural and convenient manner. Presentation of a complete specification of an econometric model should be concise, natural, general and legible.

2) Development of an appropriate scheduling algorithm to handle blocks of simultaneous equations, and inclusion of that algorithm in the MODEL processor. The scheduling algorithm should be able to affect increases in efficiency of the indicated computations based on the advantages of an automatic processor over a human in the analysis of data

dependencies among complex mathematical statements. Thorough evaluation of alternative algorithms and application of the selected algorithm to a variety of simple test cases should help establish the usefulness and correctness of the new scheduler.

3) Expansion of the code generation phase of the MODEL system to produce correct, efficient and concise implementations of object programs corresponding to program flowcharts generated by the new simultaneous block scheduling algorithm. Produced programs should be in the higher level language PL/I and should exhibit a high degree of modularity and legibility.

4) Generation and testing of a small econometric model using the newly expanded capabilities of the MODEL system. As a proof-of-principle application, the actual model of the Spanish economy used in the LINK system was converted to a mathematical representation and specified in the MODEL language. The same coefficients, historical and other data used in the LINK system was used with MODEL to reproduce LINK's results of simulations for Spain over the five year period 1978-1982. Experience gained in this proof-of-principle problem was helpful in debugging and revising some of the modifications to the system to enhance the usability of MODEL in econometrics.

5) Generation and testing of a larger econometric model. A second and more complicated national econometric model, that of France, was converted from the LINK system

implementation to a MODEL specification and tested to reproduce LINK's results for the period 1978-1982. This task helped to further smooth out any difficulties which went undetected in the preceding stage of the research, and allowed us to determine how easy large-scale use of MODEL in econometrics would be after some experience had been gained with the new system, taking into account learning curve effects noticeable in our previous development of the Spanish model. Simulation of the French model was a highly favorable experience.

6) Demonstration of the linkage of econometric models using the new MODEL system. The two models already developed and tested, Spain and France, were linked together with a simple model of the rest of the world and a novel linkage model to demonstrate the usefulness of MODEL as a general-purpose econometric modelling tool. The novel linkage model used was developed in cooperation with S. Fardoust of the University of Pennsylvania's Department of Economics and the LINK project. The linked international trade model which we thereby generated was thoroughly tested and used to satisfactorily simulate the period from 1978-1982, as were its component models before it. The success of this experiment illustrates the applicability of MODEL to econometric modelling.

All of this work was performed by the author in the University of Pennsylvania's Department of Computer and Information Science, using the facilities of the Moore

School's VAX 11/780 computer.

1.5 OUTLINE OF THE REPORT

Chapter two of this report provides a brief review of the MODEL language and processor, covering basic concepts needed to understand the algorithms and examples discussed throughout the remainder of the report.

Chapter three presents an example of the use of MODEL in econometric modelling. This should serve to firm the reader's understanding of the priniciples discussed in Chapter two and to introduce some basic concepts of econometrics used in future sections of the report.

Chapters four through six of this report present the design and implementation of modifications to the MODEL language and processor undertaken in response to modelling needs such as those discussed earlier in the report. Chapter four covers the syntax and semantics of specifying simultaneous equations in the MODEL system. Chapter five discusses modifications to the scheduling phase of the MODEL processor which handle sets of simultaneous equations. Chapter six presents the code generation techniques used to implement specifications of computations involving sets of simultaneous equations.

Chapter seven reviews the major application of the current research to date, a multimodel linked world trade model generated with the new MODEL facilities described

above. The success of this application serves to confirm the usefulness of MODEL as a tool in econometric modelling.

Chapter eight concludes the report and offers some recommendations for future research.

The four appendices contain:

i) lists of references in economics and computer science,

ii) a full listing of MODEL output reports associated with the short Spanish model,

iii) a program listing of the modifications to the MODEL scheduler, and

iv) a listing of the specification of the MODEL linked world trade model.

CHAPTER 2

REVIEW OF THE MODEL LANGUAGE AND PROCESSOR

This chapter provides a brief overview of the features and functions of the MODEL language and processor, intended to supply the reader with sufficient background to understand the balance of this report. For a more indepth treatment of these subjects, the interested reader is referred to Pnueli and Prywes (1980 and 1981).

2.1 NONPROCEDURALITY IN MODEL

MODEL is a general purpose language for specifying computational tasks. It provides users in the social, natural and engineering sciences with a very high level language for expressing computations in a familiar way which does not require expertise in computer programming. Because these users are familiar with much of classical mathematics, we have selected the language of algebraic equations as the major component of the very high level language. We refer to this language as nonprocedural. Nonprocedurality conveys the notions of a higher level and more abstract mode of

expression in the following ways:

a) It ignores all considerations of input/output and memory media of the data. Therefore there is no need to express assignments of values to memory locations. For example, the equals sign (=) in a nonprocedural language signifies algebraic equality of the two sides of an equation. In contrast, this same symbol in a procedural programming language indicates the assignment of a value to a specifically named memory location.

b) It is devoid of the notion of sequential execution of statements. The statements in a nonprocedural language may be presented in arbitrary order without changing the meaning of a specification. They are taken to be descriptive of mathematical relationships rather than prescriptive of specific actions on data.

The nonprocedural language basically consists of data descriptions and schemata, without regard to the medium on which the data are recorded or displayed, and a set of relations, typically mathematical equations, specifying the relationships among these data. To distinguish a set of nonprocedural statements from a procedural computer program, we refer to them as constituting a specification. Computational tasks are thus specified through the use of the MODEL language.

2.2 THE MODEL LANGUAGE

Statements in the MODEL language may be of three distinct types, header statements, data description statements or assertions. Data description statements describe the structure and attributes of variables participating in the specification. Assertions are equations which define the value of some variables in the specification in terms of other variables. Header statements provide documentation names for computational tasks and data aggregates in the specification.

2.2.1 DATA DESCRIPTION

Data in a MODEL specification may consist of highly structured, tree-oriented structures similar to those found in popular high level languages such as COBOL or PL/I. A structure is visualized as a tree where data constitute nodes and directed branches lead to lower level constituent substructure nodes. A complete specification of a data structure consists of a MODEL statement for each node in the underlying tree. Roughly, the syntax of a data description statement is

<data description statement> ::=

<data name> IS <node type><node description>;

<node type> ::= FILE | RECORD | GROUP | FIELD

<node description> ::= <descendants>|<data definition>

in standard BNF notation where

data name - is the reference name of a structure, i.e. the name given to a node of the tree.

node type - indicates a level in the tree. A FILE is at the root of a data tree. A FIELD is a terminal tree node representing a single variable. A RECORD is an intermediate node which is also the unit of data transfer between input/output media and memory (as in COBOL and PL/I). A GROUP is any other intermediate node in the tree structure.

descendants - supply the names and numbers of repetitions of descendant structures. If a descendant occurs only once, then the number of repetitions may be omitted. If the number of repetitions may vary, then the minimum and maximum bounds may be specified. If the number of repetitions is unknown or undetermined, this may be specified by an asterisk (*) in place of a number of repetitions. When an unknown or variable number of repetitions is indicated, the number of repetitions can be defined by an equation using the attribute qualifier SIZE. For example, if the variable is X, then its number of repetitions can be defined by equating some expression to SIZE.X.

data definition - describes a terminal node's data type, size and scale in a syntax similar to PL/I. Variables may be fixed, floating, numeric, picture, character, etc.

Data aggregates (files) may be further designated as source or target. Values of source variables are assumed to

be externally supplied, typically from input files. Values of target variables are assumed to be produced from computations as update values or external outputs to some solution file. Such a header statement is given by

<header statement> ::= <header type>:<header name>

<header type> ::= MODULE | SOURCE FILE | TARGET FILE

Note that a header statement is also used to name a MODULE, or computational task, typically one such task per input specification. The header name is a single module, source file or target file name or, in the case of source and target files, may be a list of such file names.

For example,

```
SOURCE FILE:  COEFF;
   COEFF IS FILE(CO_REC);
      CO_REC IS RECORD (C(100));
         C IS FIELD (PIC'S99.V999');
```

is the set of model statements describing a file containing variables to be used as coefficients in equations. COEFF is a file consisting of one record, CO_REC, which is composed of one hundred contiguous variables of type C, each of which is a picture variable of format S99.V999.

Another example,

```
SOURCE FILE:  TIM_SER;
   TIM_SER IS FILE (TS_REC(20));
      TS_REC IS RECORD (TS_DATA(*));
         TS_DATA IS FIELD (PIC'S999.V999');
SIZE.TS_DATA = N;
```

presents the MODEL statements which describe a file called TIM_SER, which contains historical (time series) data for use in an econometric model. There are twenty records of type TS_REC in the file, each of which contains a variable number of repetitions of the picture variable TS_DATA. The number of repetitions of the field TS_DATA is specified at runtime by the variable N, according to the relationship

```
SIZE.TS_DATA = N;
```

N is presumably defined elsewhere in the input specification.

As shown, the description of data is simple and straightforward. Each structure is listed with its constituent parts until all the data has been described.

Although data is described in MODEL statements (as in PL/I) as tree structures, we can also view data as arrays. There is a direct correspondence between the array and tree views of a data structure. Specifying a number of repetitions of a repeating descendant tree node is equivalent to specifying the range of a data vector. In general, a hierarchical data structure may be alternatively viewed as a multidimensional array, where the numbers of repetitions of nodes in the data tree give the ranges of corresponding dimensions of the multidimensional array.

Thus we may view TS_DATA in the above example as a two dimensional array. Its first, or outer, dimension corresponds to repetitions of TS_REC and its second, or inner, dimension corresponds to repetitions of TS_DATA. We

refer to the number of repetitions of a node as a size or range specification, or alternately as the size or range of the dimension. Viewing the data trees as arrays allows us to identify specific elements by using indices to indicate individual elements along corresponding dimensions. Thus

TS_DATA(n1,n2)

denotes the n2-th TS_DATA of the n1-th TS_REC. Element indices are denoted by free subscript variables that may assume integer values within the range of the appropriate dimensions. Such subscripts may be identified in MODEL specifications by simple statements of the form

<subscript declaration> ::= <subscript name> IS SUBSCRIPT;

2.2.2 ASSERTIONS

Assertions specify the transformations to be applied to data structures which have been described in data description statements. Rather than give detailed step by step instructions, as would be necessary if a procedural language were being used, the MODEL user simply identifies mathematical relationships among the variables participating in the specification, from which the MODEL processor deduces the actual execution sequences. Assertions may therefore be given in arbitrary order without changing the meaning of a specification. The variable on the left hand side of the equal sign in an assertion is the dependent variable and is defined by an expression on the right hand side of the equal sign. An expression is composed of variables and constants to which basic operators and functions have been applied.

The syntax and semantics of regular, boolean and matrix algebras are available for use in specifying such relationships. Constants, variables and boolean and arithmetic operators conform with the usage of conventional high level languages, including the ALGOL-like IF-THEN-ELSE operator whose syntax is

<variable> = IF <condition> THEN <expression_1>
ELSE <expression_2>;

a useful statement which means that if <condition> evaluates to TRUE, then <expression_ 1> defines the value of the variable; otherwise, <expression_2> is used.

An assertion statement, though similar in syntax to an assignment statement in procedural programming languages, is actually quite different. An assertion expresses the mathematical notion of equivalence between the two sides of the equal sign. That is, an assertion is an equation. This idea of assertions as equations is fundamental to the difference between procedural and nonprocedural languages. Because of the nonprocedural nature of MODEL, each variable name denotes exactly one value. The "historical" values of data, previous values overwritten by computations in procedural languages, must be explicitly represented by symbolic names. For instance, an assignment statement in a procedural language used to increment a variable within a loop might be

X = X + 1;

a statement which is meaningless in a nonprocedural language where it would be viewed as an equation. In model it would

be necessary to consider each value of X, perhaps as elements of an indexed vector, as separate and unique data entities. If successive values of X were represented as elements along a vector whose range is traversed by a subscript T, then the equivalent MODEL representation of the above assignment would be the assertion

```
X(T) = X(T-1) + 1;
```

Both dependent and independent variables should be subscripted by a list of subscript expressions corresponding to the dimensions of the variables specified in the data description statements. Subscript expressions, which may consist of any integer valued expressions, must be ordered according to the order of the dimensions of the data structure they reference. Subscripts may be omitted when doing so does not lead to ambiguity. In these cases, the MODEL processor recognizes the implication of subscripts in the expressions and fills them in automatically. Allowing omission of subscripts in certain cases can facilitate the composition of simple and correct assertions.

Qualified names, denoted by using a period (.) to connect a string of individual data names, may be used in a manner similar to PL/I to unambiguously identify specific variables when necessary. Another usage of qualified names is in specifying the range of a given array dimension, e.g. SIZE.X, as discussed above.

Assertions in a MODEL specification may contain sets of simultaneous equations, optionally nested and blocked by the

user to identify solution strategies and criteria. The MODEL processor detects such sets of simultaneous equations and generates a PL/I program to iteratively solve the system of equations and to report to the user if the system does not converge. The user may, at his option, block groups of equations together for local iterative solution and specify for each block a solution method, convergence criterion, initial data values and maximum number of iterations to be used in solving the equations. If the user omits this information, default values are assumed for these parameters and the Gauss-Seidel algorithm is used to solve the system of equations. This topic is covered in great detail throughout the remainder of this report.

2.3 THE MODEL PROCESSOR

The MODEL processor analyses a given input specification, detects any ambiguities, incompletenesses or inconsistencies in the specification and generates a PL/I program which implements the computational tasks implied or specified by that specification. Our approach is to allow the user to submit a subset of a "complete" specification, deduce the missing parts and generate corresponding statements based on implications in the portion submitted by the user. In this way some of the routine work in composing a specification may be performed by the system, and in some cases computational algorithms may be provided automatically, including the use of numerical and statistical methods as appropriate. The results of the

analysis are presented to the user in appropriate warning and error messages, and in automatically generated documentation. Messages are expressed in terms of the nonprocedural specification language, not in the procedural terms of the generated object PL/I program. Instead of demanding missing details immediately from the user, we prefer to make automatic corrections and additions wherever possible, advising the user of our actions through system messages. Even when such actions do not exactly correspond with the user's intended meaning, they often lead the user to a better understanding of his computational task so that he may complete the specification more easily.

2.3.1 THE ARRAY GRAPH

The first step in the analysis of a specification is to represent the user's statements in a convenient internal form, based on which implicit information may be derived and entered, automatic checks may be conducted and finally a schedule of program execution may be derived. We have selected an internal representation similar to Petri Nets and Data Flow Graphs, forms of directed graphs which model data dependency relationships. Our graph is a novel tool which we call an array graph. It differs from conventional methods in that each node of the graph represents the accessing or evaluating of an entire array of data rather than a single element of an array.

Each node of an array graph is potentially compound,

representing instances of the data structure or equation for all appropriate array elements. Information concerning dimensionality and range must therefore be associated with nodes of the array graph. A node which represents a data structure has subscripts associated with its source and target variables. Thus use of an array graph allows a single compound node, say A, to represent the elements from

A(1,1,...,1) to A(n1,n1,...,nm),

where n1,n2,...,nm are the ranges of the m dimensions of A from one to m respectively.

A directed edge of an array graph may similary be compound, representing all instances of dependencies among array elements at the source and destination of the edge. These dependencies imply precedence relationships in the execution of implied actions at the edge's nodes, which may take the following forms:

- hierarchical precedence refers to the need to access a source structure before its components can be accessed or the need to evaluate the components of a target structure before the aggregate structure can be stored.

- data dependency precedence refers to the need to evaluate the independent variables in an equation before the dependent variable can be evaluated; similarly, data parameters of a structure (range, length, etc.) must be evaluated before evaluating the corresponding structure.

These edges are determined from analysis of information

associated with the end nodes of the edges. Each edge must also contain information on dimensionality and range to the extent that it is compound.

Incomplete or missing information in the array graph is filled in by the MODEL processor by deducing or "propagating" essential information from user specified statements to other statements with missing information or to new statements which are generated when necessary by the system and added to the specification. Additions of information by the system include:

- dimensionality, subscripting, size of dimensions and data types of variables. When missing, these may be propagated to a variable in an equation from other variables in the equation for which these attributes have been defined.

- data description statements for variables which have been referenced in equations but have not been defined by the user.

- equations to define variables for which data descriptions have been supplied by the user. Default action is to copy variables from input to output.

- automatic inclusion of iterative solution methods when indicated by the presence of sets of simultaneous equations in the user's specification. Similarly, optimization methods may be included when there are more unknowns than equations and an objective function is

provided by the user.

In every case, the user is advised of implied actions taken by the automatic processor.

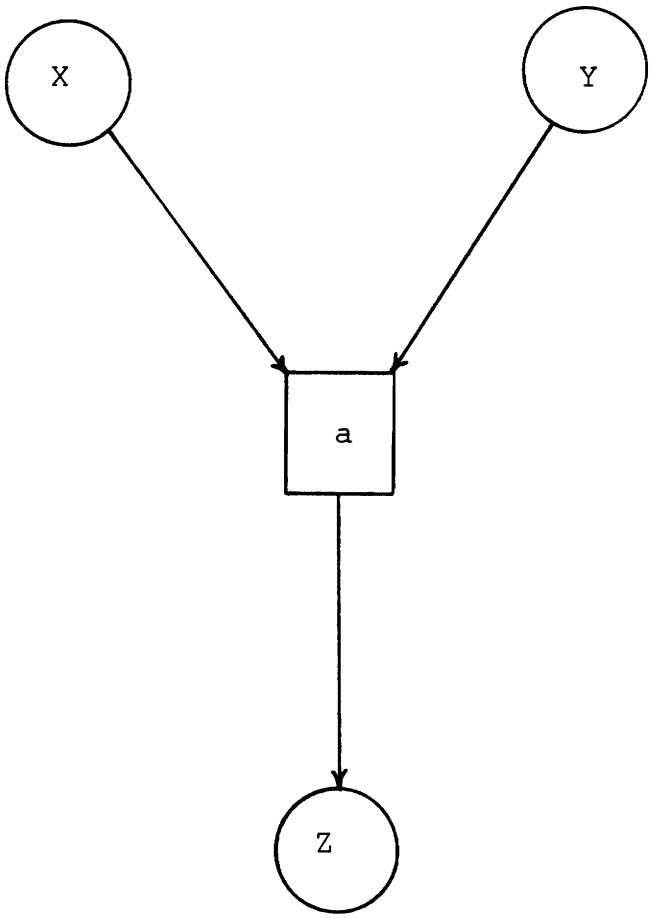
We can conveniently represent an array graph pictorially by letting a cirlce represent data structures, squares represent assertions and arrows represent directed edges. Often we associate subscript expressions with appropriate edges. Thus, the subgraph represented by the equation, named "a",

```
a: Z = 3*X + 4*Y;
```

can be visualized as shown in Figure 2-1 where the direction of the edges symbolize the direction of the dependencies. The assertions themselves, as represented by letters such as a,b,c,etc. are important components of these array graphs. As shown in Figure 2-1, the dependent variable is depicted as depending on the assertion, which in turn is dependent on the independent equation variables.

2.3.2 SCHEDULING

Scheduling of the array graph refers to generating a sequence of execution tasks which implement the computations described in the input specification. This is accomplished in a compilation process which allows us to perform extensive diagnostic analysis of the array graph. The MODEL language allows the user to choose a representation for his problem which is natural and appropriate to the situation, but which typically does not correspond with the most




```
a: Z = 3*X + 4*Y;
```

Figure 2-1. Example of a Subgraph for a Single Assertion.

efficient computation. The MODEL processor maps the user's statements into an efficient procedural computer program in the object high level language. An intermediate result of this translation is a skeletal, flowchart-like representation of the program which we call a schedule. The final, code generation phase of the system translates the individual schedule entries into efficient source code in PL/I. Consider the basic problems of translating a specification into a conventional program:

Unordered Nature of the Specification - Nonprocedurality of the language allows the user to input statements in arbitrary order. Dependency relationships in the array graph give rise to precedence constraints used to deduce correct execution sequences. An equation for defining a variable can be executed only when all the other variables on which this variable depends have been previously defined or accessed. Similarly, storage or output of a variable must be preceded by its calculation.

Handling Input/Output - User data description is independent of the storage medium, whether the data be in internal memory, external memory (secondary storage) or messages received over communication lines. The system determines, based on array graph representations of the relationships between variable and file descriptions or between variables and equations, whether the data is on an I/O device or in main memory, and schedules appropriate I/O instructions when necessary.

Analysis of Iterations - When an equation involves an array variable, translation calls for repetitive calculation of the variable for different values of the subscripts which implicitly or explicitly are included in the equation. Thus the equation must be enclosed in iterative loops, nested loops if the array is of multiple dimensions. Correct and efficient loop design must take into account many considerations, including the potentially complicated dependencies among array elements in assertions, and the possibility of sharing storage within loops for successive elements of a given array when effiency or the unmanageable size of large external data bases mandates. Iterations for numerical computation procedures, such as loops to iteratively recalculate variables in sets of simultaneous equations, must also be added when called for by analysis of the array graph dependencies.

The general approach to scheduling is to create a component graph consisting of all the Maximally Strongly Connected Components (MSCC) in the array graph and the edges connecting these MSCC's. An MSCC is either a single node of the array graph or a subgraph of the array graph containing cycles or knots of cycles caused by the existence of cyclic dependencies in the array graph. The component graph is therefore an acyclic directed graph whose component nodes may contain cycles. The component graph is then topologically sorted, resulting in a linear arrangement of components which can be regarded as a gross level representation of the flowchart. Subscripts for each

component are determined and appropriate iterations for these subscripts are added to bracket the respective components.

We can attempt to decompose multinode MSCC's into schedulable calculations by deleting certain edges when possible which may lead to unravelling of the knotted subgraphs. Consider the example of a two node MSCC given by the assertion named "a":

```
a: X(T) = IF (T=1) THEN 1 ELSE X(T-1)+1;
```

which defines the values of elements of the vector X to have the values 1,2,3,etc. If the range of the subscript T is N, i.e.

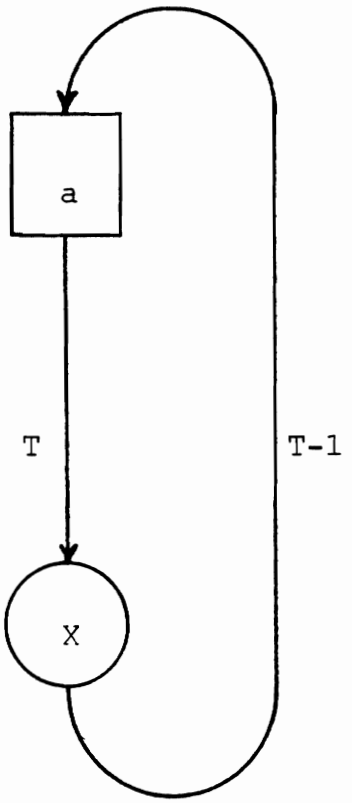
```
SIZE.X = N;
```

then a conceptual schedule of this subgraph is

```
DO T = 1 TO N;
   <subgraph consisting of nodes a and X>;
END;
```

where the array subgraph is pictorially represented as shown in Figure 2-2.

The edges in this subgraph indicate that the evaluation of the (T-1)th element of X should precede the evaluation of the Tth element. But, since we know this will occur anyway in our partial schedule owing to the monotonically increasing iteration on our loop counter T, we can delete this edge as containing extraneous information. This allows for decomposition of the MSCC and production of a detailed schedule.



a: X(T) = X(T-1) + 1;

Figure 2-2. Subgraph of a recursive assertion.

In general, to decompose an MSCC, we

1) find a dimension of the MSCC for which all of its nodes share a common subscript,

2) add an iteration around the MSCC for this subscript,

3) find edges that represent dependencies on lower valued indexed elements of the selected subscript, and

4) delete these edges to aid in decomposition of the component.

For complex MSCC's the decomposition and scheduling are performed by mutually recursive functions until all cycles which can be opened in this way have been scheduled.

Such analyses of dependencies of higher valued indexed elements on lower valued indexed elements can have important implications for the efficient use of memory. If a variable depends on only a few of the most recently calculated values of data along a given dimension, then only a few values of that variable need be in memory at any one time, and I/O for them can often be performed within the scope of the enclosing iteration. Whenever all the elements along a variable's dimension must be colocated in memory, we refer to that dimension as physical; otherwise, when only a few must be in memory at the same time as discussed above, we call that dimension virtual.

Some components of the component graph cannot be fully decomposed by the above process of deleting edges to lower

valued indexed array elements. When this occurs, there exist MSCC's consisting of cyclic dependencies which indicate the presence of sets of simultaneous equations in the input specification. In this case, an iterative numerical solution method may be applicable in lieu of the usual iteration loops generated by the scheduler. Scheduling of such numerical solution methods must take into account user statements regarding the sets of equations as well as dependencies in the array graph, including possible user specification of solution method, convergence criteria, initial data values and maximum number of iterations. The use and scheduling of simultaneous equations is the subject of much of the remainder of this report, and is discussed in considerable detail in the chapters ahead.

CHAPTER 3

MODELLING IN ECONOMETRICS

This chapter provides an example of the use of MODEL in econometric modelling. We present a description of an actual MODEL specification describing the economy of Spain, and present some background material in econometrics.

3.1 SPECIFYING ECONOMETRIC MODELS

Solution of the type of econometric models examined in this report consists of applying numerical iterative methods to sets of simultaneous equations describing the performance of certain variables in a nation's economy. These equations result from theoretical and statistical studies of the behavior of essential sectors of that economy. The equations typically involve real and monetary factors, domestic and import-export behavior, and economic and political influences. When a set of equations has been developed, it is usually in an abstract form where coefficients of the various variables are not known. Refinement of a model involves careful statistical analysis

of the sectors described in the model to verify the accuracy of the equations and to ascertain correct operational values for these coefficients. This statistical analysis is performed on historical or time-series data consisting of cross sectional data taken at periodic intervals in time. When a model and its derived coefficients can satisfactorily reproduce solution trends which correspond well with actually observed time series data, the model is deemed adequate and can be used to solve for future statistical variables which will accurately predict the future performance of that economy.

Equations in an econometric model often contain time lags as parts of the defining relationships. This concept of lags in econometrics is an important one. It recognizes the fact that economic events or trends in one sector of an economy often take a significant amount of time to be felt in other, indirectly related sectors of that economy. The solution year or period, often indicated by a subscript or index into a vector or array of time progessive solutions of an economy, can thus appear in current or lagged form. In current form, a relationship might appear as

```
X(T) = 2 * Y(T);
```

for example, meaning that for each period or year T of the solution, the economic variable X is equal to twice the value of variabe Y in that year. Lagged behavior, on the other hand, can be illustrated by the example

```
X(T) = 2 * Y(T-1);
```

which indicates that in each solution period, the value of

the economic variable X should be taken as twice the solution value of the variable Y in the previous year or period. Thus, in 1968, X is equal to two times the value which Y had in 1967.

Variables in econometric models are further classified as exogenous and endogenous variables. An exogenous variable is one whose value is considered to be determined by factors outside the scope of the theoretical bases for the given model. Time is an exogenous variable, as are congressionally determined tax rates and other values whose prediction is subject to discontinuous or uncontrollable changes. From the point of view of a local economy, import prices and export values may be considered exogenous. That is, a nation cannot determine on a theoretical basis the price that other countries will charge for goods (import prices), nor the amount of goods that other countries will wish to import from the local country (export values). Endogenous variables, on the other hand, are those which can be solved by combining the values of exogenous variables and mathematical relationships among current and lagged exogenous and endogenous variables together with numerical methods to produce periodic econometric solutions. Often such variables can be solved by iterative computer methods. Some of them can be calculated directly once others have been solved by such methods. For example, real production in different sectors, such as agriculture and manufacturing, might be solved separately in sectoral systems of interrelated equations, whereas total production in an

economy might be simply calculated by summing the individual sectors' solutions, once they have been found iteratively. It is natural, we see, to solve sets of variables based on both procedural considerations, as when iterative methods are used on various sectors, and on nonprocedural considerations, as when data dependencies are used to dictate which sectors' solutions should precede calculation of other variables.

In accordance with the above considerations, data aggregates in econometric modelling are usually arranged in some sort of vectored structure, with time representing the independent variable driving the solution. Values of the econometric variables are determined for several consecutive periods and trends among the variables are used to help determine timely and effective economic policies.

3.2 THE SHORT SPANISH MODEL

An illustration of an econometric model specified in the MODEL language is given for a simplified model of the Spanish economy. We will first take a look at the data structures and then present the defining relationships among them. The complete MODEL specification describing this model, given the MODULE name SPAIN, is presented in Figure 3-1. This is a quite simple and naive model of the Spanish economy, presented here to illustrate techniques used in more complete and realistic models.

```
MODULE: SPAIN;              /*          SPANISH TRADE MODEL          */

SOURCE FILE: SIMDEF;      /* SIMULATION PARAMETER DEFINITION FILE  */
SIMDEF IS FILE (INPUTREC);
  INPUTREC IS RECORD(BEG_YR,PD_SIM,LAG);
    BEG_YR   IS FIELD (PIC'9999');  /* YEAR OF START OF SIMULATION          */
    PD_SIM   IS FIELD (PIC'999');   /* NUMBER OF PERIODS IN OUR SIMULATION  */
    LAG      IS FIELD (PIC'999');   /* MAXIMUM NUMBER OF LAG PERIODS IN MODEL*/

SOURCE FILE:  COEFF;      /* TRADE MODEL EQUATION COEFFICIENTS FILE  */
COEFF IS FILE (CO_REC(73));
  CO_REC IS RECORD(A);
   A IS FIELD (DEC FLOAT(10));

SOURCE FILE:  TIM_SER;    /* TIME-SERIES (HISTORICAL) DATA FILE  */
TIM_SER IS FILE (TS_REC(11));
  TS_REC IS RECORD(VNAME,VNUM,NUM_PDS,TS_DATA(1:20));
    VNAME    IS FIELD (CHAR(4));     /* NAME OF MODEL VARIABLE         */
    VNUM     IS FIELD (PIC'9999');   /* NUMERIC IDENTIFIER OF VARIABLE */
    NUM_PDS  IS FIELD (PIC'9999');   /* NUMBER OF PERIODS OF TIME      */
                                     /* SERIES DATA FOR THIS VARIABLE  */
   TS_DATA IS FIELD (PIC'S99.V999'); /* TIME SERIES DATA VALUE         */
  SIZE.TS_DATA = NUM_PDS;    /* ONE DATA VALUE PER PERIOD PER VARIABLE   */

TARGET FILE: SOLUTION;   /* SIMULATION SOLUTION FILE */
SOLUTION IS FILE (SOL_GRP(*));
  SOL_GRP IS GROUP(HDR_REC,SOL_REC);   /* EACH SOLUTION GROUP CONTAINS */
                                       /*  A HEADER AND A BODY         */
  HDR_REC IS RECORD(SM_PD_ID,SM_YR_ID);
    SM_PD_ID    IS FIELD(PIC'9999');      /* SOLUTION PERIOD NUMBER  */
    SM_YR_ID  IS FIELD(PIC'BB9999');      /* SOLUTION YEAR           */

  SOL_REC IS RECORD(II,EX,GOV,IMSER,IM01,IM24,IM3,CONS,INV,IM,GDP);
   (CONS,INV,II,EX,IM,GOV,GDP,IMSER,IM01,IM24,IM3) ARE FIELD (PIC'BB99.V(6)9');

  SIZE.SOL_GRP = PD_SIM;      /* ONE SOLUTION RECORD FOR EACH PD. OF SIM. */
  T IS SUBSCRIPT;             /* T (TIME) IS A COUNTER OF SIMULATION PDS. */

/* NOW WE DEFINE VALUES FOR EXOGENOUS VARIABLES IN OUR MODEL  */

II   (T) = TS_DATA( 3,T);  EX   (T) = TS_DATA( 4,T);  GOV  (T) = TS_DATA( 6,T);
IMSER(T) = TS_DATA( 8,T);  IM01 (T) = TS_DATA( 9,T);  IM24 (T) = TS_DATA(10,T);
IM3  (T) = TS_DATA(11,T);

/* HERE, WE DEFINE VALUES AMONG OUR ENDOGENOUS VARIABLES  */

CONS(T)  = IF (T>LAG) THEN A(1) + A(2)*GDP(T) + A(3)*CONS(T-1)
           ELSE           TS_DATA(1,T);
INV(T)   = IF (T>LAG) THEN A(4) + A(5)*GDP(T) + A(6)*GDP(T-1) + II(T)
           ELSE           TS_DATA(2,T);
IM(T)    = IF (T>LAG) THEN IM01(T)+IM24(T)+IM3(T)+A(61)+A(62)*GDP(T)+IMSER(T)
           ELSE           TS_DATA(5,T);
GDP(T)   = IF (T>LAG) THEN CONS(T) + INV(T) + EX(T) + GOV(T) - IM(T)
           ELSE           TS_DATA(7,T);

/* FINALLY, WE PRESENT EQUATIONS FOR A SIMPLE SOLUTION GROUP HEADER */

SM_PD_ID(T) = T;                      /* NUMBER OF SOLUTION PERIOD  */
SM_YR_ID(T) = BEG_YR + T - 1;         /* YEAR OF SIMULATION RESULTS */
```

Figure 3-1. MODEL Specification of the Short Spanish Model.

There are four data files associated with SPAIN: three source (input) files, SIMDEF, COEFF and TIM_SER; and one target (output) file, SOLUTION.

The SIMDEF or SIMulation DEFinition file contains certain parameters used to control the simulation. After the object PL/I program has been generated and compiled, this is the file which the user manipulates to run different simulations. It consists of a single record INPUTREC containing three picture variable fields, BEG_YR, PD_SIM and LAG. BEG_YR is the beginning year or period of the simulation, PD_SIM is the desired number of periods to be solved in our simulation, and LAG indicates the maximum number of lagged periods in the model equations. When the resultant Spanish sumulation is executed, the solutions of the economy will be generated for PD_SIM years beginning with year BEG_YR. LAG is the number of historical data periods which must be read into memory before beginning the solution procedure.

COEFF is a file containing statistically derived coefficients of the model equations. It consists of seventy-three records, each called CO_REC and each containing a single floating point variable A(i). The A(i)'s may be referenced using a single subscript corresponding to their positions in the input file and their numbers in the model equations. Thus A(7) is the seventh number in the coefficient file and the coefficient A7 in the mathematical representation of the econometric model.

TIM_SER is a file containing historical (time series) data of the Spanish economy. There are eleven records named TS_REC in this file, each containing information about a single one of the eleven variables participating in the econometric model. Each TS_REC contains three simple fields called VNAME, VNUM and NUM_PDS, and a vector of historical data called TS_DATA for the corresponding variable. VNAME is simply the documentation name of the variable corresponding to that TS_REC; VNUM is a similar documentation identification number associated with the variable (numbered consecutively from one to eleven in the respective TS_REC's); and TS_DATA contains from one to twenty historical values of the variable. The exact number of periods for which historical data is available is specified by the remaining field, NUM_PDS. This relation is stated in the assertion

SIZE.TS_DATA = NUM_PDS;

For uniformity, we constrain all TS_DATA vectors to begin in the same year, but they may end in different years depending upon the availability of data for the particular economic variable. A particular historical value for a variable thus has two indices or subscripts corresponding to the two dimensions of the data array, one dimension for the repetitions of TS_REC and one for the repetitions within TS_DATA. Thus TS_DATA(4,7) refers to the seventh time series datum in the fourth record of the file. More generally, TS_DATA(I,J) refers to the Jth time series datum in the Ith record of the file, where I and J are subscript

variables in the model specification.

The remaining file in our specification, SOLUTION, is the output file which will contain solutions for the various years of our dynamic simulation. This is a virtual file (indicated by the * in SOL_GRP(*)) which contains a variable number of solution groups, each named SOL_GRP and each describing the solution of the Spanish economy for a single year or period. The size of the SOL_GRP array, and hence the number of solutions to be found, is given by the assertion

SIZE.SOL_GRP = PD_SIM;

where PD_SIM has been described in the SIMDEF file above. Each SOL_GRP contains two records, HDR_REC and SOL_REC, each of which correspond to a single line (record) in the SOLUTION file. The first line in each SOL_GRP is a header record HDR_REC which contains the fields SIM_PD_ID and SM_YR_ID, the period number and year of our dynamic simulation, respectively. The second line in each SOL_GRP is a solution record SOL_REC which gives a list of picture variable solution values for variables in the simulation, one solution value per variable. These variables are named as fields in SOL_REC and have the following meaning in the Spanish economy:

CONS - consumption

INV - total investment

II - autonomous government investment

EX - total exports

IM - total imports

GOV - total government expenditures

GDP - gross domestic product

IMSER- imported services

IMO1,

IM24,

IM3 - imports in various commodity categories

These variables can be thought of as virtual arrays of data, one array element per SOL_GRP. Thus, CONS(T), for example, represents the value of the consumption variable in solution period T. T, representing time as a solution period counter, is identified as a SUBSCRIPT variable in the input specification.

Now that we have defined our data in data description statements, we must present the assertions, or defining relationships among the data. Below, and in the input specification, we have grouped these statements into three logically sensible sections defining the exogenous model variables, endogenous variables and solution header variables.

Exogenous Variables - These assertions allow us to copy the historical values of solution variables into the appropriate solution variables. As noted, exogenous values

are supplied from sources outside the econometric model. All we need do is relate the appropriate historical data array to the corresponding solution variable. A typical assertion defining an exogenous variable is

```
GOV(T) = TS_DATA(6,T);
```

which defines values of GOV, the government expenditures variable. This assertion says that in each solution period, we take the value of GOV from the corresponding data element in the sixth historical data record. This sixth record is the one containing time series data for GOV. In this simple model, no theoretical relationships are given to determine GOV if it were not thus given exogenously. If the simulation is extended into the future, then "future historical data" for exogenous variables must be extrapolated using outside sources and these values must be made available in the TIM_SER file.

Endogenous Variables - These assertions describe the mathematical relationships among the economic variable in defining equations to be solved by numerical methods. The MODEL system will generate an iterative solution program to determine the values of variables found to be involved in simultaneous sets of equations. The method of solution is the subject of much of the remainder of this report. Assertions here have the form

```
X(T) = IF (T>LAG) THEN f(A(i),other vars,lagged vars)
       ELSE TS_DATA(n,T);
```

where X represents a given variable to be defined, n is the index of that variable's time series data record in the

TIM_SER file, and f is a functional relationship defining X in terms of other current and lagged variables and the equation coefficients A(i). In early solution periods, when T is less than or equal to LAG as defined in the simulation definition file, solution values are merely copied from historical data. This ensures that enough values of the solution variables will be established so that when T is greater than LAG, "turning on" the equation, references to lagged solution variables will be satisfiable. If such initial copying were not done, then for example, CONS(T-1) when T=1 would be meaningless.

Header Records - These two assertions simply define the relationships among the solution period counter T and the year and period of the simulation. If BEG_YR is 1980, for example, then in the first three years of the simulation SM_PD_ID is 1,2,3 and SIM_YR_ID is 1980,1981,1982. These values help to identify individual solutions in the output file and thereby make it more readable.

When the specification of the Spanish model is submitted to the MODEL processor, the results are a correct and efficient PL/I program which implements the described econometric model, a complete set of documentation reporting on various aspects of the specification and its object PL/I program, and thorough warning and error messages, where appropriate, to the user. When compiled, linked and executed, the PL/I program reads the appropriate source files, solves the econometric model according to the

equations provided and reports to the user solutions of the model.

Although the example of Spain given here is a simple and naive one, it serves well to illustrate the methods used to solve more complicated and general econometric models.

3.3 LINKAGE OF NATIONAL ECONOMETRIC MODELS

In the example of the short Spanish model presented above, we showed how certain trade variables were determined exogenously to a given economy's model. The value of a nation's exports and the price of its imports cannot be dictated by factors internal to its economy, but are rather decided by policies and actions of the external countries with which it trades. The given country, in turn, helps determine the value of exports and price of imports of its trading partners. Linkage of econometric models entails replacing these exogenous trade variables with the actual values of the variables as determined by co-executed models of international trading partners. Iterative solution of several models proceeds simultaneously to solve the system of equations linking the export variables of each country to the import variables of its trading partners. Iterations continue until convergence criteria indicate that each country's internal and trade sectors have converged. At this point the trade model is said to have converged and the values of all the model variables represent a solution for that period.

Figure 3-2 shows a block diagram of a simple system of two nations, A and B, which are each other's exclusive trading partners. In this case, country A determines the price of imports and value of exports of country B, and country B determines the price of imports and value of exports of country A. Thus the system of equations linking the trading partners, i.e. the linkage model is simply

A.VX = B.VM;

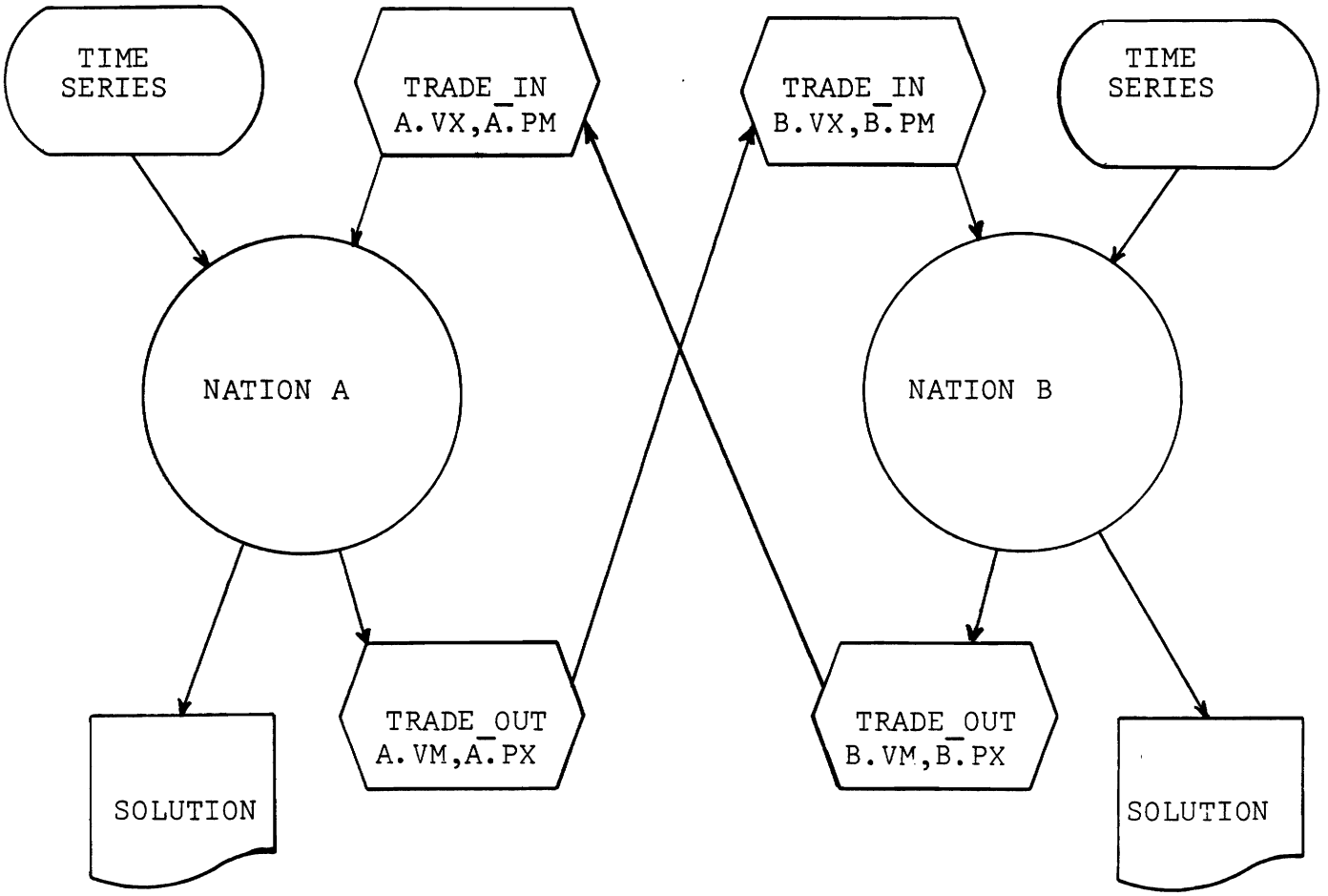
A.PM = B.PX;

B.VX = A.VM;

B.PM = A.PX;

where PX = price of exports, PM = price of imports, VX = value of exports, VM = value of imports and "A." and "B." serve as qualifiers to identify the nation whose variable we mean.

Note that the trade variables can be thought of as being outputs of a linkage model, and the needed trade variables can be thought of as being inputs to a linkage model. In the dynamic simulations which are our main concern, such linkage models are of prime importance. During the development stage of an international trade model, however, national econometric models are likely to be generated independently. In order to test such models, we can simply replace the required trade variables, which are endogenous in a linked configuration, with exogenous values obtained from input files as in the preceding chapter's example of Spain. One criterion for design of econometric modelling systems is to allow such switches from local to



A.VX = B.VM
A.PM = B.PX
B.VX = A.VM
B.PM = A.PX

Figure 3-2. Block Diagram of a Two-Nation Trade Relationship.

linked computing modes to take place quickly and easily. In Figure 3-2, we can visualize the models as running either in linked mode, using the four equation linkage model presented above, or in local mode, where the TRADE_IN and TRADE_OUT values are in external data files. In the latter case input files provide estimated or extrapolated values based on analyses of historical data, and are replaced in the former case by real values of dynamically simulated trade.

The simple linkage model described above is insufficient when trade models of greater scope and complexity are used. Addition of more models representing countries or regions with a complex network of trade in many goods and services requires more general and extensible methods of linkage. Gana, Hickman, Lau and Jacobson (1979) have investigated alternative approaches to linkage of national econometric models. In Project LINK, an analytical tool known as a trade share matrix (TSM) is used to determine the proportions of individual models' imports as a share of its trading partners' exports. Aggregate trade variables can be appropriately disaggregated by commodity groups and routed to the correct trading nation's economy. The trade share matrix and accompanying mathematical relationships for aggregating/disaggregating trade variables and for routing commodities and services among nations/regions comprise a linkage model. The linkage model and the individual models of participating local economies together comprise the international trade model. A block diagram of such a system is shown is Figure 3-3, where A-Z

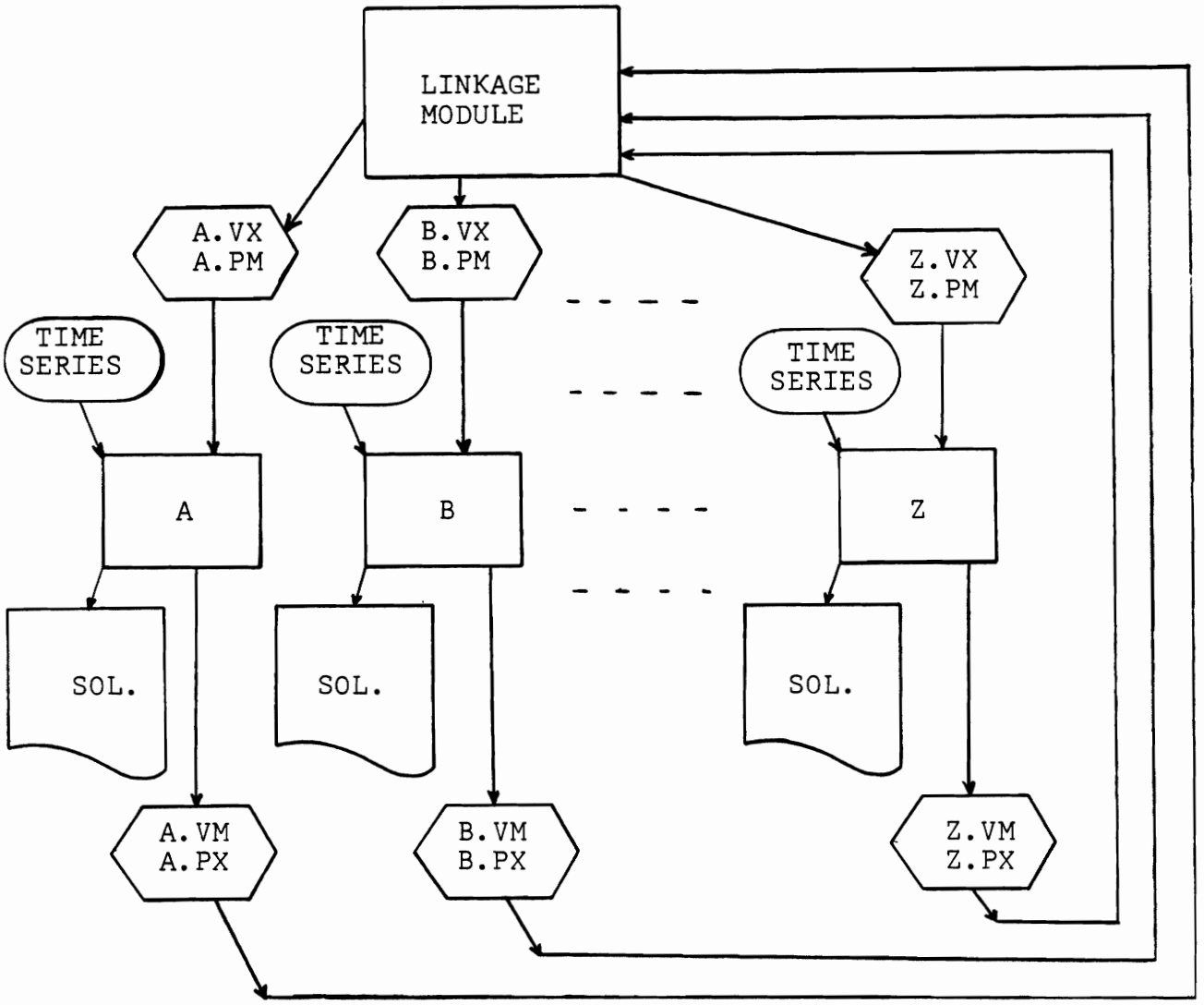


Figure 3-3. Block Diagram of a Multinational Trade Model Using a Central Linkage Model.

represent individual country or region econometric models.

CHAPTER 4

SIMULTANEOUS EQUATIONS: SYNTAX AND SEMANTICS

In this chapter we describe the nature and method of identification of simultaneous equations in the MODEL system. The role of the user versus the automatic processor is discussed. User control of scheduling and solution generation are presented using the tools of the MODEL language. The syntax and semantics of specifying computations involving sets of simultaneous equations are presented.

4.1 SIMULTANEOUS EQUATIONS IN MODEL

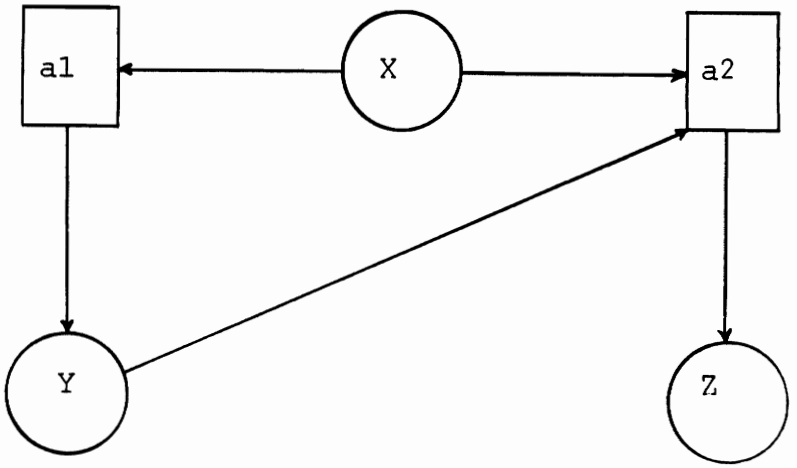
In the scheduling phase of the MODEL processor, the array graph is topologically sorted to determine a sequence of nodes dependent on one another in strict order. When this order, based on the direction of data dependency edges in the graph, has been determined, loops may be added around subgraphs of the sorted graph to satisfy the range and dimensionality aspects of the appropriate nodes. For example, consider the assertions:

```
a1:  Y = SIN(X);
a2:  Z = (X**2 + Y**2)**0.5;
a3:  X = 1;
```

A simplified representation of the array graph for this partial specification (which excludes data and file definitions) is given in Figure 4-1.

During scheduling, the corresponding dependency-sorted graph can be visualized as shown in Figure 4-2a or, more simply as shown in Figure 4-2b where, in the second case we have removed the edge from X to a2, which became redundant when overall topological sorting guaranteed the order of these nodes via other paths. We say that the graph has now been decomposed into a sequence of single nodes, each connected to the next by a single dependency relationship. The integrity of our calculation of data values for X, Y and Z in the order specified by this new decomposed graph has been guaranteed by the sorting process. If X, Y and Z have non-zero dimensionality, an iteration loop may be easily added to enclose the necessary nodes of the sorted graph, and we would proceed with code generation.

In a large class of problems, we may not be able to fully decompose a graph into a linear list of individual assertion and data nodes. Rather, the presence of cyclical dependencies, or cycles, in the array graph may limit our sorting process to the production of a graph whose nodes are non-trivial subgraphs. That is the nodes of the sorted graph may no longer be simple assertion or data nodes, but




```
a1: Y = SIN(X);
a2: Z = (X**2 + Y**2)**0.5;
a3: X = 1;
```

Figure 4-1. Topologically sortable subgraph of three equations.

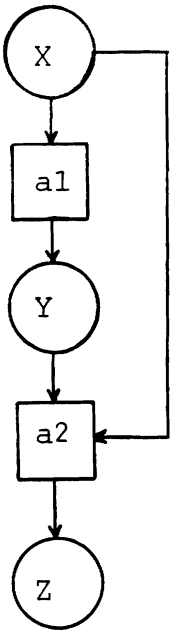


Figure 4-2a. Subgraph of Figure 4-1, Shown Sorted Topologically

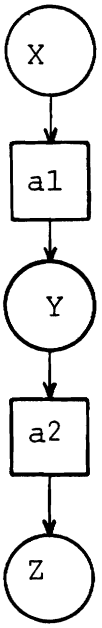


Figure 4-2b. Subgraph of Figure 4-1, Sorted and with Redundant Edge from X to a2 Removed.

may contain one or more assertion and data nodes connected by cyclical dependencies. A cyclical dependency occurs when the value of some datum depends not solely on the values of other data, but also, directly or indirectly, on its own value. Some of these cycles can be further decomposed and scheduled by the process of deletion of edges from higher valued indexed to lower valued indexed elements of a dimension of some data structure as described in Chapter two of this report. In this chapter, and throughout the remainder of this report, we will be concerned with cycles where this type of decomposition is not possible. These cycles indicate the presence of simultaneous equations in the user specification.

An example of a simple cyclical dependency can be illustrated by the single assertion

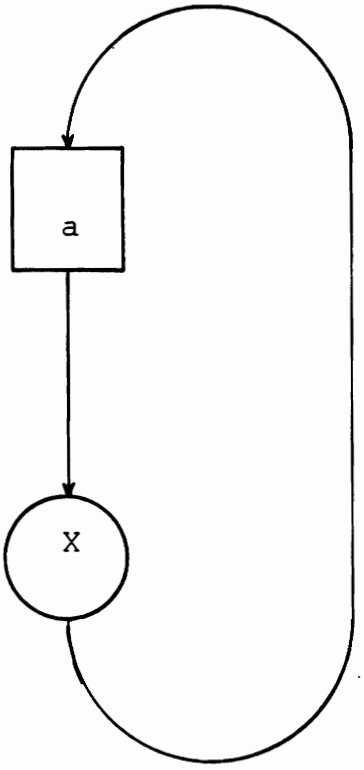
```
a: X = 3*(X**2);
```

This is a simple case where the assertion for X is directly self-referential, representing the mathematical relationship

```
X = 1/3;
```

The array graph for this example is simply as shown in Figure 4-3. We see that, beginning at X, we may traverse a set of edges of the graph and end up back at X again. This circular path is a cycle or cyclical dependency. Of course, more than one assertion may be involved in the cycle. Consider the assertions

```
a1:  X = Y + Z + 1;
a2:  Y = X + Z - 11;
a3:  Z = X + Y - 9;
```




```
a: X = 3*(X**2);
```

Figure 4-3. Subgraph of a Self-Referential Assertion.

representing a set of three simultaneous equations in three variables with solution

$$(x,y,z) = (10,4,5).$$

Here we will find that the corresponding graph contains several cycles as shown in Figure 4-4. Here each datum in the graph depends on every other variable involved. It is impossible to decompose this graph on the basis of data dependencies alone, because every node is topologically equivalent to every other node in the array graph. Therefore, scheduling cannot proceed in a linear fashion as in the usual case, and a new strategy must be adopted to deal with simultaneous equations.

When the array graph representation of the specification contains only linearly sortable dependencies, the task of the MODEL system's scheduling and code generation phases is simply to order these assertions in correspondence with constraints implied by the data dependencies and generate code to correspond with the described calculations, with properly added iteration loops to ensure that the full range of every datum's dimensions are correctly accounted for. While this is no trivial task, all the mathematical information describing the relationships among source and target variables are explicitly or implicitly expressed in the specification in the form of data dependencies.

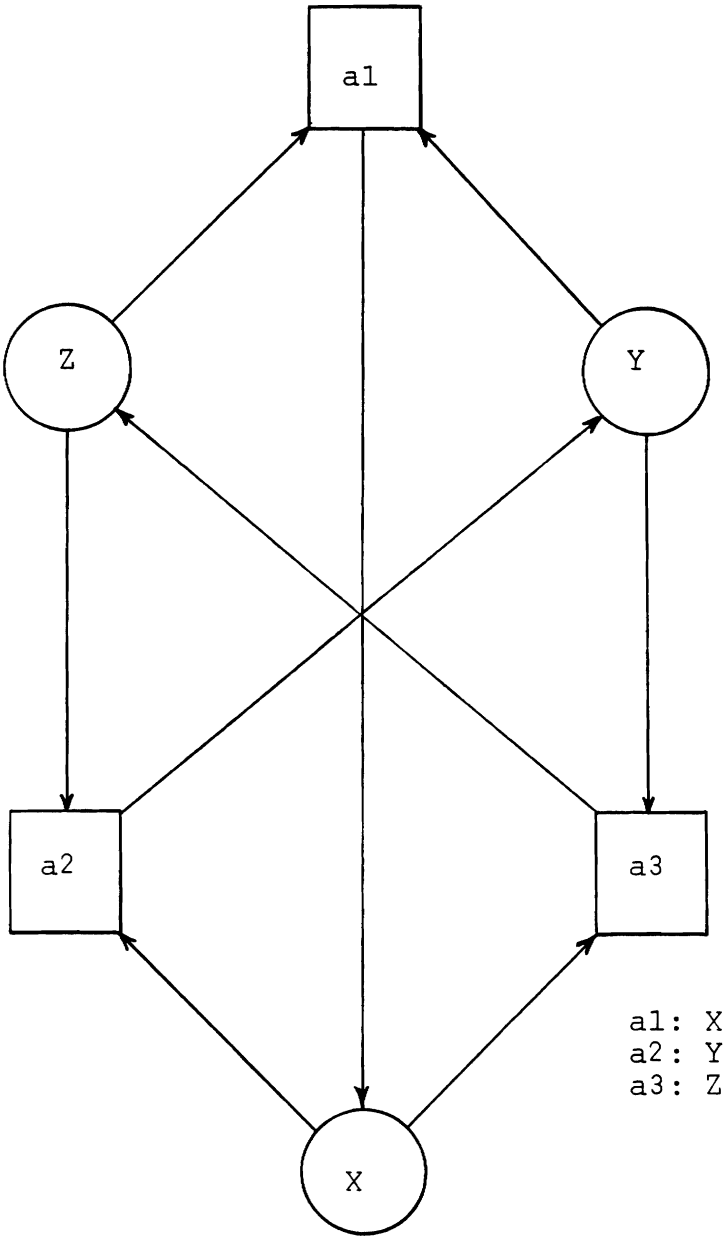


Figure 4-4. Subgraph of a Set of Three Simultaneous Equations Containing Several Cycles

4.2 ITERATIVE SOLUTION METHODS IN MODEL

When the input specification contains simultaneous equations, however, we have seen that insufficient information exists in the array graph to determine the order of calculation of data values by the same method of examination of data dependencies. In such cases, the objective of the MODEL processor is not simply to order the assertions to ensure correct precedence of calculations, but to generate a correct object program which provides a solution of the given simultaneous equations. Thus it is necessary to provide instructions to the MODEL system as to how to solve the simultaneous equations. One way to do this would be to force the user to include a great deal more information in his input specification, detailing the manner in which he wishes the simultaneous equations to be solved. This would be a great burden on the modeler and is one of the major drawbacks of many currently used methods which the current research is intended to remedy. The other way to provide such information, and the way chosen in the current implementation, is to build into the MODEL system the necessary expertise in computer programming and numerical analysis for generating solutions to such problems on its own.

A number of well known algorithms exist in the field of numerical analysis for determining solutions to sets of simultaneous equations. It is our intention that the user shall eventually be able to select from among several of

these algorithms for use in solving his systems of equations, and that the MODEL processor will implement an object program according to such indications, or according to criteria applied automatically to determine the optimum solution method for the particular probem at hand. In this stage of the research, we have implemented only one such solution method to serve as a basis for study of the effectiveness of the new MODEL system as a tool in modelling with simultaneous equations. The algorithm selected for implementation is the Gauss-Seidel iterative method, one of the most well known, straightforward and general such methods in numerical analysis.

In this method, initial values are assigned to each of the variables in the set of equations to be solved. We then iteratively sequence through the equations, calculating a new value for every variable in the set of equations in each iteration. In each equation, the current values of independent variables, as determined by their most recent iterative recalculation, are used to calculate a new replacement value for the dependent variable. This iteration process is repeated until either

1) the difference between the current and previous values for all independent variables does not exceed a given percentage, in which case the set of equations is said to have converged and the current values constitute a solution of the set of equations, or

2) some prescribed maximum number of iterations

has been reached without convergence, in which case we say that the set of simultaneous equations has diverged.

For a more complete and theoretical explanation of the Gauss-Seidel solution method the reader is referred to almost any good text in numerical analysis.

Thus we may infer that, in order to generate a solving program for a set of simultaneous equation, we must supply the following additional information to the MODEL system:

1) solution method - We take this always to be the Gauss-Seidel method in the current implementation, but other algorithms exist and could be implemented in the future.

2) maximum number of iterations - A maximum number of iterations to perform before assuming the set of equations has no solution and terminating the iteration process.

3) convergence criterion - The relative distance between successively calculated values of independent variables which, when not exceeded, signals convergence of the equations.

4) initial values - Optional initial values to use as starting points in the iterative solution process. The closeness of initial values to the eventual solution values can reduce the number of iteration required for convergence.

5) recalculation order - The order desired by the user for recalculating the values of new

independent variables during each iteration of the solution process. Changes in the order of recalculation can also reduce the number of iterations required for convergence of the system of equations.

In our implementation, we have provided new features in the MODEL language to allow the user to supply this information to the extent he wishes, while building sufficient intelligence into the MODEL system to enable the user to omit any or all of the information and have the MODEL system generate a solution on its own.

4.3 SEMANTICS OF SIMULTANEOUS BLOCKS

The semantic tool we have supplied to provide this additional information is the BLOCK statement. When a user wishes to solve a set of simultaneous equations, he simply places assertions representing these equations in his program specification and encloses the set of assertions by a BLOCK-END statement pair. This indicates to the MODEL processor that any simultaneous equations detected within the scope of the matching BLOCK and END statements shall be solved by an iterative solution method, with the order of recalculation of dependent variables to correspond with the order of their defining assertions in the input specification. Returning to our earlier example, a user might place the following statements in his input specification:

```
BLOCK SOLVE_IT;
    Y = X**2 + Z;
    Z = Y + 1;
    X = 1;
END SOLVE_IT;
```

By enclosing these equations within a BLOCK-END pair, the user indicates his desire to generate a program solving the simultaneous equations by an iterative procedure. He also requests the system to generate code which recalculates data values in the order Y,Z,X.

For documentation purposes, the user must include a name for any BLOCK specification in his input file. In the above case, the name SOLVE_IT was chosen. Ostensibly, users will choose descriptive titles to enhance the informative quality of reports generated by the MODEL system.

The intent of the BLOCK statement is descriptive rather than prescriptive. That is, the BLOCK specificatins do not convey strict scheduling instructions to the MODEL processor, but are intended to provide clues and guidelines to be used in generating solution procedures. This is consistent with the general philosophy of the MODEL system, relieving the user of the responsibility for procedural explicitness and its requisite high degree of programming proficiency. In keeping with this philosophy, there is in effect a "default BLOCK" specification, an invisible BLOCK-END pair which encloses the entire input specification. The name of this default block is

$MAINBLOCK. Any simultaneous equations found within the scope of this default block, i.e. any such equations in the input file at all, will be solved by the system fully automatically even if the user omits all additional block information. Thus, the modeler can create simple Gauss-Seidel solution procedures simply by inserting a set of simultaneous equations in his specification. If he does not wish to alter the default system block descriptors, he need do nothing else. The MODEL system will independently generate a correct and efficient solution procedure for him fully automatically.

Thus the precise meaning of the BLOCK specification is that the preparer of the specification suspects that there may be cyclical dependencies among the assertions between the BLOCK and END statement pair, and that he desires to have these equations solved locally, within a possibly larger number of simultaneous equations. Further, if such dependencies are found, then the system is to take the order of the assertions in the specification as a guideline for scheduling the assertions in the corresponding topologically unsortable component subgraph to produce a solving iterative procedure. This meaning of the BLOCK statement is adopted because when large or complicated sets of equations are involved, the user may be unsure about the exact nature of the data dependencies in his specification. Rather than perform exhaustive (and exhausting) mathematical analyses of the equations himself, the user may wish to submit the entire group to the MODEL processor within a BLOCK-END pair

and have the system automatically determine which equations are involved in the simultaneous equations and which are not, based on the data dependencies in the corresponding array graph. Those equations which do not participate in the actual simultaneous set can be straightforwardly calculated in a solution program, while those that do will be solved in a nested local iterative solution procedure, as part of a possibly larger iterative solution involving other equations or blocks.

Among those which do form simultaneous equations, the MODEL system will, to the extent possible, rely on dependencies for scheduling determinants, but when stymied by topologically unsortable subgraphs it must rely on ordering guidelines established by the user in his preparation of the input file. Although the user may not have performed lengthy analyses of the dependencies among his equations, we assume that he is sufficiently familiar with the nature of his model to suggest a guideline for the order of iterative recalculations of his variables. Speed of convergence may be greatly affected by the order of the equations, and in fact the Gauss-Seidel algorithm may converge or diverge based solely on the order of the equations in the iteration procedure. Strategies for choosing an optimal order of equations when employing the Gauss-Seidel algorithm exist but depend on the sizes and relative values of the coefficients of the model equations. Since, as a rule, the equation coefficients will not be available until runtime of the generated solution program,

so that the user alone and not the MODEL processor might have knowledge of their values at the time of compilation of the specification, we must accept the user's order as a guideline for scheduling when topological decomposition of the array graph is blocked by the presence of cycles.

As will be shown in a later chapter of this report, edges of the array graph which conflict with the order suggested by the user are selectively deleted and the resultant subgraph is resubmitted to the topological decomposition process. This is continued until a fully decomposed graph is achieved and a complete flowchart has been constructed. Because deletion of one edge may release several nodes of the subgraph from a given cycle, the final order of calculations is partly dependent on the user's clues and partly dependent on the inherent data dependencies in the system of equations. Although the exact order of the scheduled nodes may therefore differ from the exact order in the input specification, as a result of data dependencies peculiar to any given system of equations, such differences will affect neither the correctness nor the speed of convergence of the generated solution program. In fact, by removing from the scope of the iterative solution assertions which the user mistakenly or unknowingly included in his BLOCK, the MODEL system is able to increase the efficiency of the solution procedure. This is because recalculation of variables which dependency analysis has revealed are not actually a part of the cycle of equations need not be iteratively performed. Rather, the values of such variables

need be calculated only once, either before or after the iterative solution loop, as dependencies dictate.

4.4 NESTING OF SIMULTANEOUS BLOCKS

We may describe a set of simultaneous equations in the notation of matrix algegra,

$$A\ X = B$$

where the matrices are given as in Figure 4-5. Here, X represents the variables to be solved in our eventual Gauss-Seidel solution procedure, and A is a matrix of coefficients of the variables in the system of simultaneous equations. The matrix B contains constant terms of the equations. In real applications, when the number of variables to be solved is more than a few, it is likely that the matrix of coefficients A is a sparse matrix. That is, A will contain a large proportion of zero-valued coefficients. This is because in large mathematical models, most of the model variables are directly dependent on only a few of the remaining variables in the set of equations. In such cases the user may wish to further improve the efficiency of his program by requesting that his overall iterative solution procedure contain smaller, localized iterations over groups of variables whose coefficients form dense partitions of the overall coefficient matrix for his model. The user may wish to make such a request whenever his knowledge of the behavior of the system of equations he is modelling suggests that some subgroups of equations exhibit tightly coupled behavior. This is not limited to equations with nonzero

$$\sum_{j=1}^{n} a_{ij}x_j = b_i \qquad i=1,\ldots,n$$

$$A\ X = B$$

$$A = \left[a_{ij}\right]$$

$$X^T = (x_1,\ldots,x_n)$$

$$B^T = (b_1,\ldots,b_n)$$

T = matrix transpose operation

Figure 4-5. Matrix Representation of Simultaneous Equations.

coefficient matrices, but may occur whenever the functional relationships among subgroups of variables exhibits tightly coupled behavior. For example, an econometrician modelling the world economy would request localized iterative solution of groups of equations representing individual countries' economies, on the basis that the tight coupling among variables internal to each country renders them more efficiently solved in local iterative procedures.

In the MODEL system, we call this localization of iterations nesting. To request a nested solution, the user simply encloses subgroups of equations in local or nested BLOCK-END statement pairs. This nesting appears similar to the normal nesting of iterations in a procedural computer program, except that in the nonprocedural context of the MODEL system it provides information about target variable coupling rather than indicating a specific series of operations. If we symbolically represent assertion by ai, bi and ci, then

```
BLOCK WORLD;
    c1;
    c2;
  BLOCK A_COUNTRY;
        a1;
        a2;
    END A_COUNTRY;
    BLOCK B_COUNTRY;
        b1;
        b2;
    END B_COUNTRY;
END WORLD;
```

represents a conceptual model of the world economy using BLOCK notation, where countries A and B are solved by local iterations within the global iteration for the total WORLD

model. In this example, ai represent assertions describing the economy of country A, bi represent assertions describing the economy of country B, and ci represent assertions describing the linkage model of trade relationships among countries A and B.

Figure 4-6 depicts a procedural analog of the nonprocedural view of nesting presented above, illustrating how individual country models are solved in local iterations within global iterations of an international linkage model.

4.5 SYNTAX OF BLOCK STATEMENTS: BLOCK DESCRIPTORS

As we noted earlier, certain other solution procedure characteristics besides calculation order may be specified by the modeler. These are solution method, relative error and maximum iterations. Choice of solution method allows the user to select from among a number of possible procedures for numerical calculation of solution to his sets of simultaneous equations. Relative error refers to the convergence criterion for equations within a given block, the maximum distance between successive iteration values of variables in that block for convergence to be satisfied. Maximum iterations is the maximum allowable number of iterations of a block of equations before it is assumed that the system of equations has diverged. For each BLOCK in his input specification, the user may optionally employ simple English-like statements for supplying the values of these block descriptors. These are appended to the BLOCK

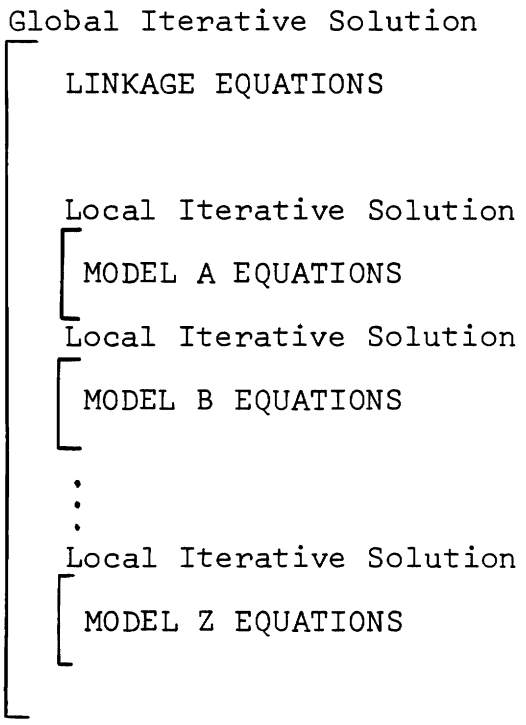


Figure 4-6. Procedural Nesting of Iterative Solutions of Blocks of Equations.

statement in any order, separated by commas as in

```
BLOCK SOLVER:  SOLUTION IS GAUSS_SEIDEL,
               RELATIVE ERROR IS x,
               MAXIMUM ITERATION IS n;
```

Here x is a real number and n is an integer. Allowable abbreviations of the above descriptors are SOL for SOLUTION, REL for RELATIVE, ERR for ERROR, MAX for MAXIMUM and ITER for ITERATION. Default values for any of the above descriptors are supplied automatically by the system for use when the user omits the corresponding descriptors from his input specification. These default values are:

DESCRIPTOR	DEFAULT
SOLUTION	GAUSS_SEIDEL
REL ERROR	0.0001, i.e. 1/100 %
MAX ITER	100

4.6 THE INITIAL STATEMENT

We have noted that the user may optionally supply values to be used as initial values for variables which are solved by means of iterative procedures. He may do so by use of the INITIAL statement, as for instance

```
INITIAL.VAR_NAME = <expression>;
```

which nonprocedurally supplies an initial value for the variable VAR_NAME, to be used if VAR_NAME is found to belong to a system of simultaneous equations. In the absence of user supplied initial values, the MODEL processor employs a process of choosing initial values based on the following

policy:

- if the variable is of dimension zero (i.e. a scalar), then we choose the constant value one for an initial value
- if the variable is of dimension greater than one, then along each dimension we set the first element's initial value equal to the constant value one, and each succeeding element's initial value equal to the solution value of its immediate predecessor element along that dimension

This policy was developed after careful analysis of modelers' data bases in a broad range of potential applications. The choice when dimension is greater than unity improves programming efficiency in many real world cases where solutions of variables along a given dimension are often not very distant from one another, as for example when variables in an economy are solved for several successive years and exhibit slowly changing values. Of course, the user can override such default values through the use of the INITIAL statement as described above.

4.7 THE END STATEMENT

The user may, as described above, employ a BLOCK statement to indicate the beginning of a block of simultaneous equations. Similarly, he may signify the end of a given block by providing a matching END statement. Semantically, an end statement may be written as

END;

or

END <block name>;

where the former statement indicates an end to the current innermost open block and the latter indicates an end to the block whose name is given following the word END. The MODEL system will provide end statements wherever they are missing, by adding END statements for unmatched BLOCK statements to the end of the input specification. The order of addition of END statements corresponds to the order of the unmatched BLOCK statements in the specification. That is, open blocks are closed in the same order that they were opened in the corresponding BLOCK statements. The default system block is closed last so that this invisible block encloses the user's entire input specification. Although this policy has proven quite effective, it can sometimes lead to nesting structures different from the intent of the modeler. For this reason, it is recommended that the user include END statements in all but the simplest nesting situations. This avoids the ambiguity of unclosed BLOCKs.

In the next chapter of this report we will examine how the MODEL scheduler uses the BLOCK descriptors to help determine a schedule of computations from the user's specification.

CHAPTER 5

SIMULTANEOUS EQUATIONS: SCHEDULING

In this chapter, we examine in detail the scheduling of array graphs which contain cyclical dependencies. Some examples of input specifications describing systems of simultaneous equations and their corresponding generated flowcharts are presented.

5.1 THE NEW BLOCK SCHEDULER

The bulk of the processing during the scheduling phase of the MODEL processor is performed by two mutually recursive procedures, SCHEDULE_COMPONENT and SCHEDULE_GRAPH. Both procedures accept as an argument a pointer to an array subgraph which is to be scheduled. SCHEDULE_GRAPH decomposes the array graph into maximally strongly connected components (MSCC's) and passes each to a separate instance of SCHEDULE_COMPONENT, which generates flowcharts for each such component of the array graph. These small flowcharts, one for each MSCC, when concatenated together, comprise the flowchart for the entire array graph. A maximally strongly

connected component (MSCC) is defined as a subgraph where there is a path between any pair of member nodes. If the MSCC is a simple data or assertion node, then it can be scheduled as the next event in the object program flowchart. When the MSCC is a set of interconnected nodes, however, possibly indicating the presence of a set of simultaneous equations, then the previously employed algorithms are insufficient and we must employ a new method of creating elements in the flowchart. In this chapter, we are solely concerned with MSCC's which cannot be further decomposed and which therefore form sets of simultaneous equations.

We continue to use these same two procedures, SCHEDULE_GRAPH and SCHEDULE_COMPONENT. A third procedure, SIMUL_BLK (for SIMultaneous BLocK scheduler), has been added to SCHEDULE_COMPONENT to handle the cases where there are systems of simultaneous equations in the input specification. SIMUL_BLK analyses an MSCC and, on the basis of ordering considerations supplied by the user through his use of BLOCK-END statement pairs in the input specification, selectively deletes edges from the MSCC, generates flowchart structures to begin simultaneous solution procedures, and resubmits the array graph to the SCHEDULE_GRAPH procedure. This selective edge removal enables SCHEDULE_GRAPH to further decompose the MSCC into two or more smaller MSCC's, each of which is similarly submitted to SCHEDULE_COMPONENT for further scheduling. This process of selective edge deletion and resubmission of the resultant subgraphs continues until every MSCC is the size of a single node, at

which time the flowchart for the entire object program will be complete.

Several scheduling algorithms for scheduling MSCC's were discussed and evaluated during the development of the current MODEL system, and two were implemented and tested before the current final version was chosen. Two central ideas were used as guidelines in the development and selection of this algorithm, in keeping with the evolution to date of the MODEL system:

1) the MODEL processor should perform as much scheduling and analysis on its own as is reasonably possible, without need for intervention by the user, and

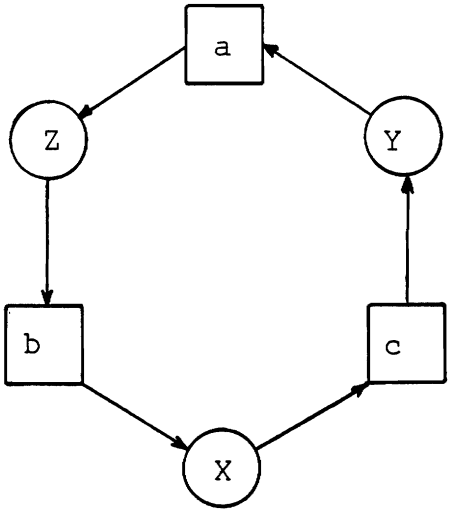
2) the scheduler should evaluate the array graph as represented by the user in the input specification and should add as little extra information as is reasonably possible.

A tradeoff exists between the actions of requiring completeness in the input specification and building intelligence into the MODEL processor. Requiring the user to include much detailed information in his input specification may make the MODEL processor more difficult to use, discouraging its goal of attractiveness to those who are unskilled in computer science. Building in a great deal of intelligence is difficult except in very specific areas. Our desire to keep the MODEL system a general purpose software tool precludes the addition of scheduling

algorithms aimed at very specific end applications.

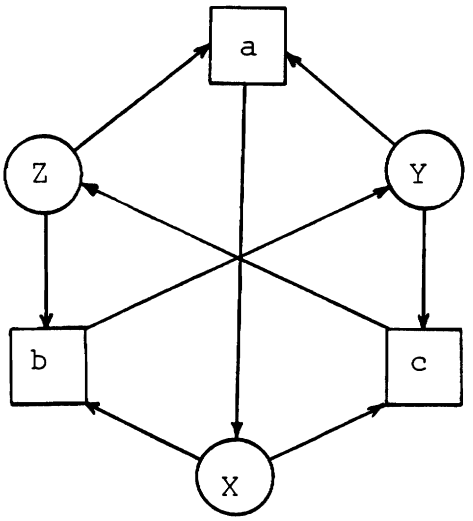
In the current version, we have applied these ideas by limiting the amount and type of manipulation which SIMUL_BLK may perform on the underlying array graph. We have built in enough intelligence to make the MODEL system a powerful tool in modelling systems of simultaneous equations. We have also limited its manipulation of the array graph to the selective deletion of edges, so that the possibility of adding contradicting or limiting information to the input model is eliminated.

In keeping with this philosophy, we decided to retain in our new algorithm the basic approach of topological sorting based on data dependencies. By definition, a multinode MSCC is not further sortable. However, the deletion of one or more edges from the MSCC may well render it further decomposable, although to what extent depends upon the particular graph's dependencies. In the simple example in Figure 5-1a, removal of a single edge renders the entire graph sortable. But in the next graph, in Figure 5-1b, removal of the indicated edge allows only partial sorting of the graph, leaving another smaller MSCC still present. Although these examples are simple in order to illustrate the principles of edge deletion, the method is general and can easily be extended to very large and complex graphs. Removal of even a single edge can often greatly unravel a complicated graph and allow further topological decomposition.




```
a: Z = Y;
b: X = Z;
c: Y = X;
```

Figure 5-1a. Subgraph where Removal of an Edge Allows Sorting.




```
a: X = Y + Z;
b: Y = X + Z;
c: Z = X + Y;
```

Figure 5-1b. Subgraph where Removal of an Edge Allows Partial Sorting.

It is possible to analyze a given graph and determine which edges, when removed, will give rise to the greatest amount of such unravelling. Such analysis would reduce the number of required steps in the entire submit-analyze-delete-resubmit process, hastening the production of our flowchart. This approach is not open to us when generating iterative solution procedures because the order of nodes in the generated flowchart must depend not only on relative dependencies, but also on the strict order implied by the user in his input specification. Recall that the speed of convergence, and in fact the very stability of the system of simultaneous equations, may depend on the order of iterative recalculation. Therefore, while we must remove edges to allow further decomposition of the array graph, we may only delete edges which contradict or interfere with the order of assertions implied by the user in the input specification.

Recall also that the philosophy of the MODEL system is to promote maximum efficiency in the modelling of simultaneous equations, to which end we wish wherever possible to rearrange portions of the user's specification. We wish to move assertions presented in inner nested blocks to outer levels of nesting whenever data dependencies indicate this is possible, in order to avoid unnecesarry recalculation of locally irrelevant variables. If a variable in an inner block does not depend simultaneously on other variables in that block, it is possible to move it to the next outer level of nesting, improving program

efficiency without adversely affecting correctness of the computations.

In keeping with these considerations, the following approach was developed to choose which edges should be deleted by the SIMUL_BLK procedure. When an MSCC graph is presented for scheduling, it is examined both from the data dependency point of view and the specification order point of view. The first component of the MSCC, from a specification order viewpoint, is identified. Then all edges emanating from other nodes in the MSCC and terminating at this first component are removed. This eliminates data dependencies which deter scheduling of the first component, and permits it to be unravelled from the rest of the MSCC, consistent with both specification order and its own internal data dependencies.

Besides simply removing edges, it is the responsibility of the simultaneous block scheduler to generate some flowchart data structures in order to indicate the beginning and end of new simultaneous solution iterations in the generated object program. Whenever the first component of the MSCC is a single node and the innermost block to which it belongs has not been started in the flowchart, SIMUL_BLK generates the data structures required to indicate a simultaneous solution iteration in the program flowchart. It then makes a note of which block it has started, and after deleting the proper edges as above, passes the newly simplified subgraph recursively to SCHEDULE_GRAPH for

further action. The annotation of which blocks have or have not been started in the flowchart ensures that indicated solution iterations will not be redundantly generated.

5.2 DETAILED PRESENTATION OF THE SIMUL_BLK PROCEDURE

In the following discussion we examine in detail the internal data structures and processing of the simultaneous block scheduler, SIMUL_BLK. Some familiarity with the data usage in the MODEL processor will be assumed. For more in-depth background material, see Pnueli, Lu and Prywes (1980). See Appendix I for a program listing of SIMUL_BLK.

5.2.1 DATA STRUCTURES OF SIMUL_BLK

There are five primary data structures which the simultaneous block scheduler manipulates, the SIM structure, the BLOCK structure, the EDGE structure, the GNODE structure and the flowchart structure. Of these data structures, only the SIM structure is generated and used exclusively within SIMUL_BLK. The other data structures have more general uses in the scheduling phase and throughout the MODEL processor. Each of these data structures is described below:

1. The GNODE Structure - The GNODE structure is the scheduling phase's simplified internal representation of array graphs and subgraphs, illustrated in Figure 5-2. It omits some information which is present in the complete graph and which is unnecessary for scheduling purposes, and is thereby smaller and more straightforward to manipulate.

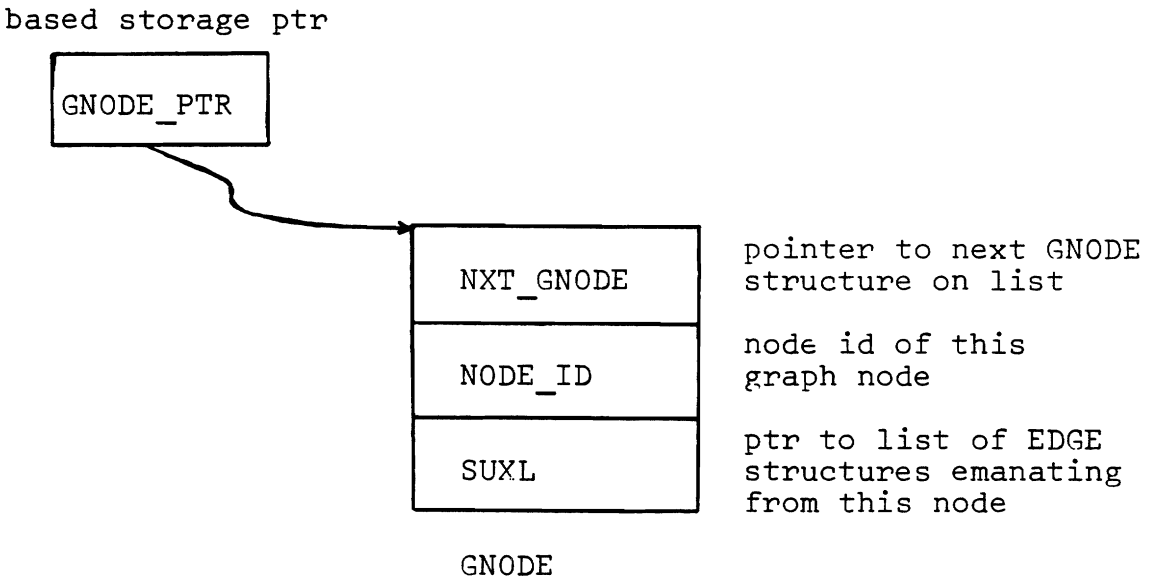


Figure 5-2. Diagram of the GNODE structure.

This data structure is a linked list of data structures, each of type GNODE and referred to by the based pointer GNODE_PTR. Each structure instance on the list represents a node of the array graph and contains the following skeletal data necessary to arrive at the graph in its entirety:

NODE_ID - the node identification number of the current node

SUXL - a pointer to a list of edge structures which emanate from the current node and terminate in its "successors"

NXT_GNODE - a pointer to the next GNODE structure on the list

Prior to scheduling, a list of this type is constructed and passed to the scheduling routine SCHEDULE_GRAPH via the pointer GRAPH, which points to the beginning of the entire GNODE list. As scheduling proceeds and some of the nodes are scheduled in the object program flowchart, the scheduled nodes are removed from the GNODE list. Each new, smaller GNODE list is passed to a new recursion of SCHEDULE_GRAPH, which proceeds with the scheduling of that subgraph as if it were an entirely new and complete graph. When scheduling is completed, the GNODE list will have shrunk to a single node submitted to the final instance of SCHEDULE_GRAPH, whose scheduling task is at last trivial. At this point the complete solution flowchart will have been generated and the task of the scheduling phase over.

The GNODE list is used by SIMUL_BLK to examine the array graph from the point of view of data dependencies.

When SCHEDULE_GRAPH finds that the subgraph it has been passed comprises an MSCC, indicating that it cannot proceed any further on the basis of dependency data alone, the subgraph is passed to SIMUL_BLK for further action. To the dependency information, SIMUL_BLK adds its knowledge of input file order and user specified BLOCK statements, and can break the apparent scheduling deadlock by scheduling an iterative solution and selective deletion of edges. The deletion of these edges is accomplished through manipulation of the list of edges to successor nodes, SUXL, associated with each GNODE structure.

2. The Flowchart Structure - A flowchart structure, called locally SELMNT by SIMUL_BLK, is generated and added to the growing object program flowchart list whenever SIMUL_BLK finds it necessary to begin an iterative solution procedure. This structure is illustrated in Figure 5-3. The data included in this structure straightforwardly provides the information required to generate PL/I object code for the iterative procedure, including the maximum number of iterations, the relative error convergence criterion, the type of solution method (always Gauss-Seidel for the present), etc. Another piece of data included is a pointer to a SIMUL_BLK-generated list of data fields which require initialization before the solution iterations can proceed. These initial values are gathered by the code generation phase from associative memory, where they are stored in INITIAL.X statements supplied by the user, or by default values generated by the MODEL processor. SIMUL_BLK

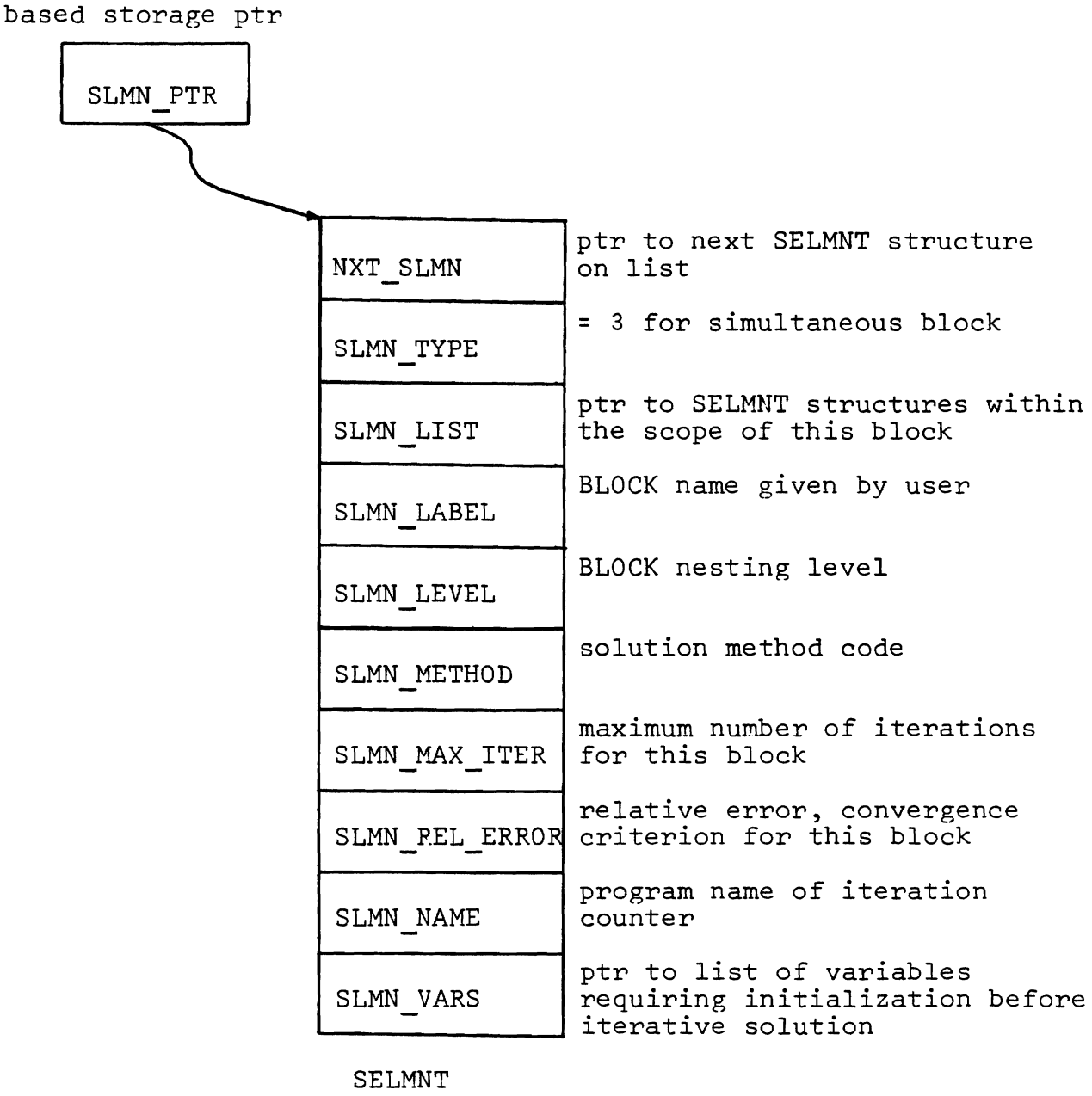


Figure 5-3. Diagram of the flowchart structure.

manipulation of SELMNT is limited to concatenation of new flowchart data to the object program flowchart.

3. The BLOCK Structure - The BLOCK structure list, illustrated in Figure 5-4, provides information which describes the specification's nesting structure as provided by the user through his use of BLOCK-END statement pairs. During the syntax analysis phase of the MODEL processor, each BLOCK statement provided by the user gives rise to a structure on this doubly linked list. Optionally supplied user details of relative error, maximum iterations, etc., are included in the BLOCK structure. Default values are filled in during syntax analysis wherever the user omits this data. In each BLOCK structure, the system also provides a nesting level number, a down-list pointer, an up-list pointer and a pointer to the parent block of each block to assist SIMUL_BLK in manipulating the BLOCK list.

When SIMUL_BLK determines that the first assertion to be scheduled (i.e. unravelled by deleting incoming edges from assertions later in the input specification) is part of a BLOCK which has not yet been generated, SIMUL_BLK allocates a SELMNT flowchart structure, fills in the necessary information provided by the corresponding BLOCK structure, and concatenates the new flowchart structure onto the existing object program flowchart. This new flowchart structure will signify the beginning of an iterative solution procedure to later code generation phases of the MODEL system. At the time this flowchart structure is

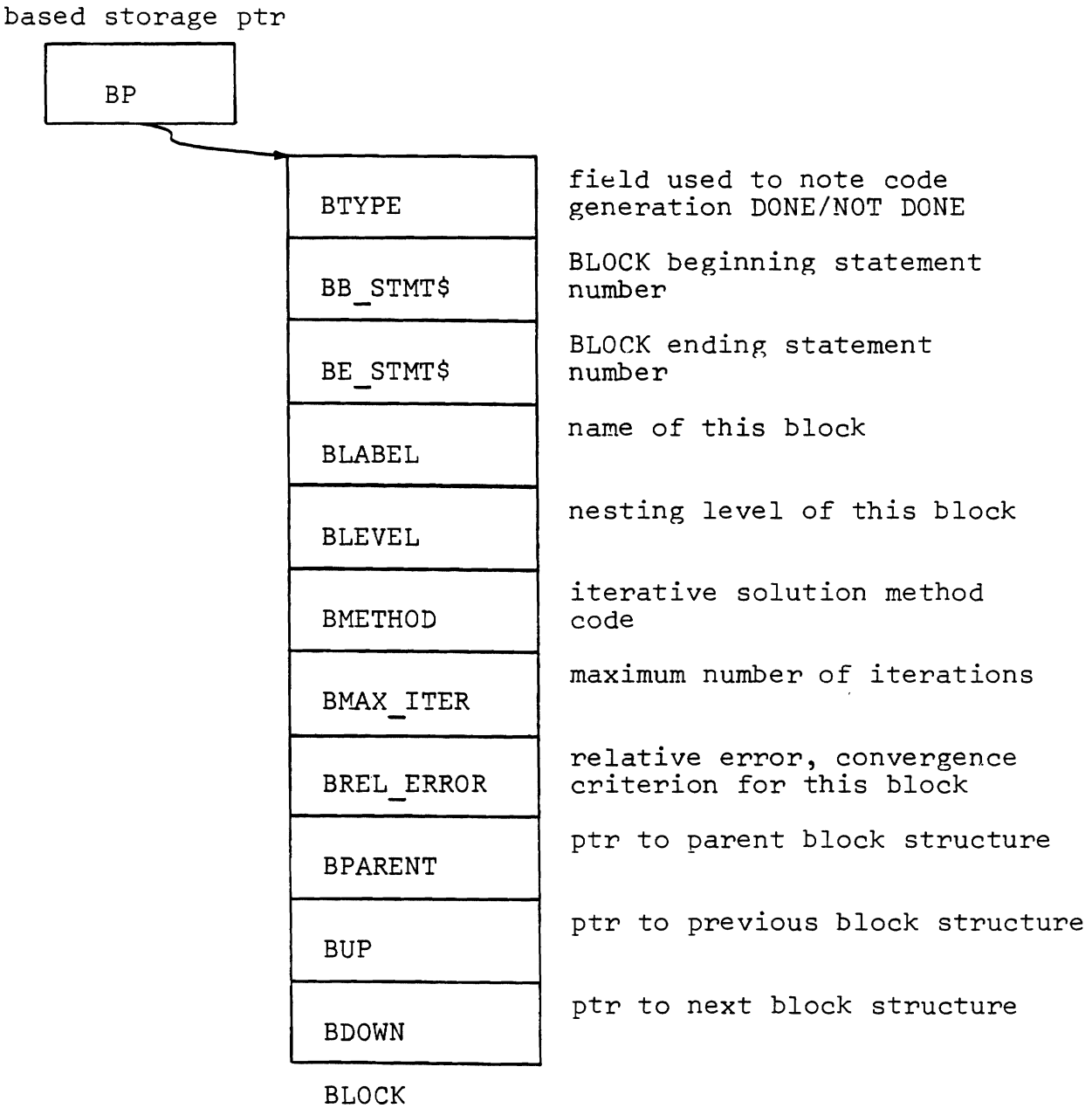


Figure 5-4. Diagram of the BLOCK structure.

generated, SIMUL_BLK sets the field BTYPE in the corresponding BLOCK structure equal to 'DONE', an indication to future recursive instances of SIMUL_BLK that the required iteration procedure has been started. At the end of scheduling, when recursive descendants of this instance of SIMUL_BLK return to it, SIMUL_BLK resets the BTYPE field to spaces before returning itself to its calling program.

Associated with each BLOCK is a beginning and ending statement number, given by BB_STMT$ and BE_STMT$, respectively, in its corresponding BLOCK structure. Both these data are used by SIMUL_BLK to help determine the first component in the MSCC it is attempting to unravel. If SIMUL_BLK finds that the first specification order assertion in the MSCC belongs in a block of equations which has not been completely scheduled, then the first component consists of all assertions in the MSCC whose innermost containing block is the same as the innermost containing block of the first assertion. Otherwise, the first component consists of just the first assertion by itself. Identification of the first component is essential, because unravelling of the MSCC is accomplished by deletion of all edges emanating from later assertions in the MSCC and terminating in the first component.

4. The EDGE Structure - There is a list of EDGE structures associated with each node in the subgraph, and pointed to by the SUXL pointer in each node's GNODE structure list entry. The EDGE structure, illustrated in

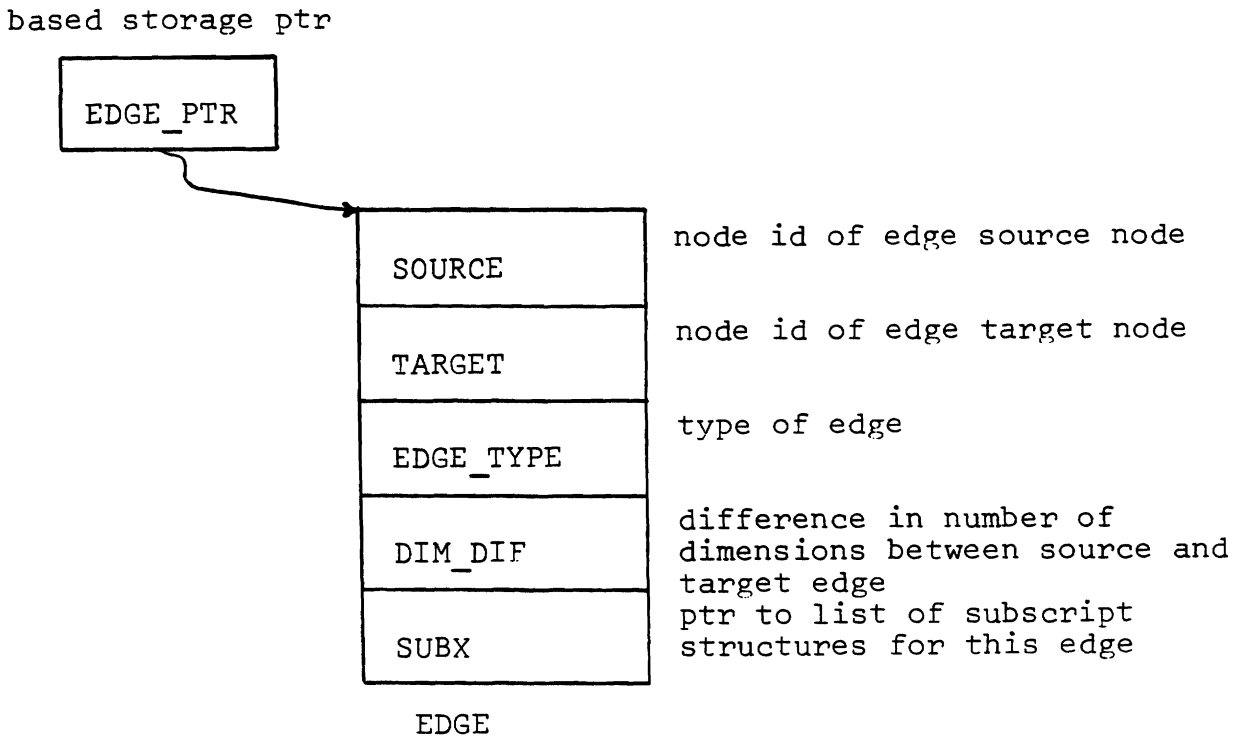


Figure 5-5. Diagram of the EDGE structure.

Figure 5-5, allows access to range, dimensionality and other important data detailing the characteristics of edges emanating from the individual graph nodes. Two important data fields in the EDGE structure are SOURCE and TARGET, integer variables representing the node id's of the graph nodes where the given edge originates and terminates, respectively.

When SIMUL_BLK has arrived at a list of nodes comprising the first component in the MSCC, the edge structure lists of all later specification order nodes are examined, and any EDGE structure whose TARGET field bears the node id of a node in the first component is deleted. This deletion is accomplished simply by manipulating the pointers of the EDGE structure lists and freeing the memory occupied by each based storage EDGE structure.

5. The SIM Structure - The SIM structure shown in Figure 5-6, is the main working data structure of the SIMUL_BLK procedure. It is a list of data structures, one for each node in the MSCC, maintained in input file order.

The GNODE representation of the array subgraph is in essentially random order from the point of view of the sequence of assertions in the user's input specification. As we have seen, however, this sequence of assertions is essential if we are to unravel an array graph with cyclical dependencies consistent with the user's specification. To bridge this gap of input sequence information, and to assist in the deletion of selected edges, SIMUL_BLK manipulates the

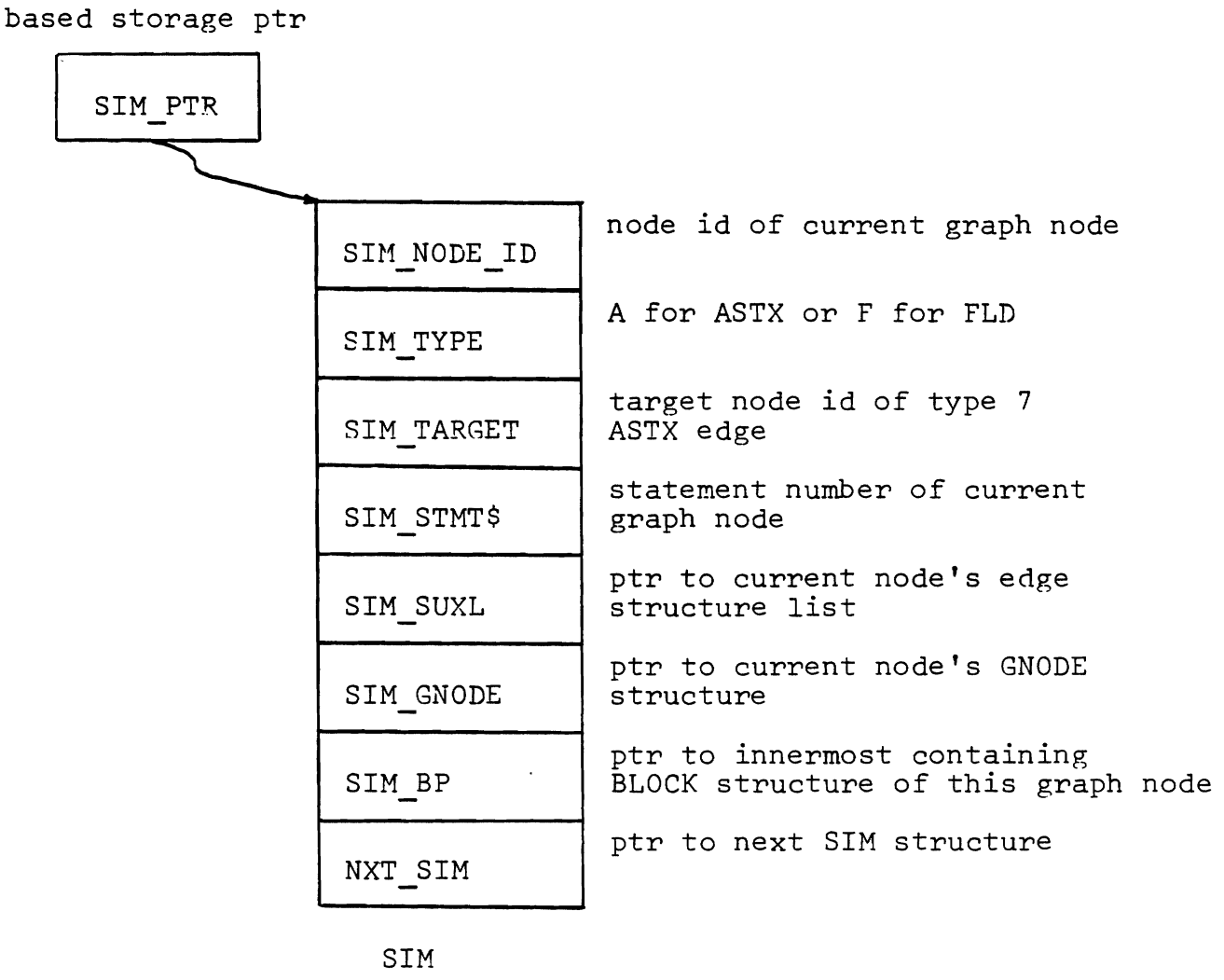


Figure 5-6. Diagram of the SIM structure.

SIM structure.

Each SIM structure contains the following data:

SIM_NODE_ID - identification code of the corresponding array graph node

SIM_TARGET - target node id of this node, if the node is an assertion

SIM_STMT$ - statement number of assertion node

SIM_GNODE - cross-reference pointer to this node's GNODE structure

NXT_SIM - pointer to the next SIM in the input file order list

There are two types of nodes in the MSCC's which SIMUL_BLK is asked to unravel, FLD (field) type nodes and ASTX (assertion) type nodes. That is, the MSCC contains only assertions and the fields which are their targets. Since the actual statement number in associative memory corresponding to a FLD type node is the statement number of its data description statement, this statement number is irrelevant for our purposes. This is because we are concerned with specification order based on assertion order, not data description order. Thus for the purposes of SIMUL_BLK we consider both the data field and its defining assertion to have the same statement number, equal to the actual statement number of the assertion (ASTX) statement. In our linear ordering of SIM structures, we conventionally place the structure for FLD nodes immediately following the structure for their defining ASTX nodes. The purpose of the SIM_TARGET field in the SIM structure is to enable us to

correctly place FLD type nodes after their defining ASTX nodes, of which they are targets.

In the following discussion we describe the processing which takes place in SIMUL_BLK and examine the manipulation of the basic data structures described above.

5.2.2 PROCESSING IN SIMUL_BLK

As do the mutually recursive procedures SCHEDULE_GRAPH and SCHEDULE_COMPONENT, SIMUL_BLK returns a pointer to a flowchart list generated as a result of SIMUL_BLK's and its descendants' activities. The processing which takes place in SIMUL_BLK is logically divided into nine separate sequential tasks. The actions of each of these nine tasks are summarized below. For a complete program listing of the SIMUL_BLK procedure, see Appendix I of this report.

The purpose of each task in SIMUL_BLK can be briefly identified by the actual comments which precede these tasks in the PL/I source code:

1. initialize the list of SIM structures with hol and eol members
2. construct a SIM data structure for each ASTX node in specification order
3. construct a SIM data structure for each FLD target of the preceding ASTX elements
4. find the last SIM structure in the first component
5. make a list of node id's in the first

component

6. optionally print diagnostic/status information

7. delete all backward edges into the first component

8. find the smallest block which contains our entire MSCC

9. allocate block if necessary and recursively schedule or terminate

These nine tasks are described in more detail below:

1. Initialize the list of SIM structures with hol and eol members.

To assist in subsequent manipulation of the SIM structure list, the linked list is initially created with two dummy members, a head-of-list (hol) member and an end-of-list (eol) member. They are given statement numbers which clearly identify them as external to our real input specification, and are linked via hol's NXT_SIM pointer.

2. Construct a SIM data structure for each ASTX in specification order

Here we search through the GNODE structure list which describes the array subgraph to be scheduled and create a SIM structure for each assertion type (ASTX) node found, adding the SIM structure to our growing list of SIM structures in the appropriate specification order. The key by which these assertion SIM's are ordered is a sequence number created during syntax analysis which is given by

sequence number = 10 * line number + indentation.

Here indentation refers to the offset (counted from zero) of the given assertion from the beginning of the given line. For example, an assertion name associatively stored as 'AASS413' refers to the fourth assertion on line forty-one of the specification. We will alternately refer to this coded sequence number and simply a statement number in the following paragraphs.

For each SIM structure allocated in this search, we find its statement number and, by searching through its EDGE list for a type seven edge, we determine the node id of the FLD type data node which is defined by this assertion. This FLD node id will be used to help us correctly add the FLD type SIM structures to out list in the next processing task. Next, we search through the growing SIM list and insert our new structure just after the last assertion whose statement number is less than the current node's statement number, thereby generating the list of SIM's in specification order.

As we construct our SIM list, we compare the statement numbers of the assertions with the two variables FIRST_STMT$ and LAST_STMT$, updating them as necessary so that they reflect the statement numbers of the first and the last assertions in our MSCC from a specification viewpoint, respectively. These variables will be used later to help identify the first component in our MSCC.

3. Construct a SIM data structure for each FLD target of the preceding ASTX elements.

Here we search once again through the GNODE list and this

time create a SIM structure for each FLD type node found. We place these into the SIM list immediately following the assertions which define the fields' values. Thus for each newly allocated FLD type SIM structure, we search the SIM list for the ASTX type SIM strucure whose SIM_TARGET field is equal to the node id of the given FLD, and insert the new SIM structure immediately after it on the SIM list.

At the completion of this task we have completed the SIM structure list. Every assertion node from the array subgraph is represented by one SIM structure, in order of increasing statement numbers on the list. Similarly, every data field node is represented by one SIM structure, which is located on the SIM list immediately following the SIM of the assertion node which defines this data field.

4. Find the last SIM structure in the first component. As discussed earlier, we will attempt to unravel the first component from the the array subgraph by selectively deleting edges into the first component from later assertions in the input specification. Recall that the first component in this context may be either a single node or a nested block of nodes, depending on the particular array subgraph at hand. Here we look through the BLOCK structure list, the head of which is pointed to by a variable called BHOL (for Block Head Of List) which was set during syntax analysis when the BLOCK list was created. IF we find a block whose scope begins earlier than FIRST_STMT$ and ends within the given MSCC, then the first component

consists of all those nodes within our MSCC which are within the scope of that first block. Otherwise, our first component consists of the very first assertion in our MSCC and its associated FLD type target node. In either case, we set a pointer to the SIM structure immediately after the last SIM structure in the first component in our MSCC, to be used in further processing.

5. Make a list of node id's in the first component. In this task we simply traverse the SIM list from its beginning up to the end of the first component in the list, found in the preceding task. As we do this, we construct a list of corresponding structures, each of which contains simply the node id of its SIM list structure. We now have a linked list of node id's in the first component of our MSCC.

6. Optionally print diagnostic/status information. The MODEL system user can select at runtime a diagnostic output feature triggered by the binary flag TRACE. If TRACE is set, desired phases of the MODEL processor print out diagnostic and/or debugging information which is extremely useful for software development of MODEL modifications. Here, as elsewhere in MODEL, certain diagnostic output commands are left in the source code even after the containing procedure has been debugged and verified. This policy is an invaluable aid when modification and extension of existing MODEL software becomes necessary. Here, the diagnostic information consists of a description of edges in the array graph underlying our current MSCC.

7. Delete all backward edges into the first component. Beginning at the first SIM structure after the first component in our MSCC, we traverse the rest of the SIM list. For each one, we use the embedded pointer to the node's corresponding GNODE structure to retrieve a pointer (GNODE.SUXL) to a list of edges emanating from that node. We then look through this list of edges and for each edge, we look through the list of nodes in our first component found in task five above. Whenever we find an edge whose target node matches a node id in the list of nodes, we delete that edge by manipulating its EDGE list linkage pointers. This simple process enables us to unravel the MSCC somewhat so that successive attempts at scheduling some of its nodes will be successful.

8. Find the smallest block which contains our entire MSCC.

This task is accomplished simply by looking through the BLOCK structure list, beginning at its head of list, until we find a block whose scope encompasses our entire MSCC. That is, we find a block whose BLOCK statement number is less than FIRST_STMT$ and whose END statement number is greater than LAST_STMT$. This block will be used in the next task.

9. Allocate block if necessary and recursively schedule or terminate.

If the BLOCK found in task eight above has not been started in our solution program flowchart, then we must create a

flowchart data structure, enter into it the proper data concerning this block's iterative solution procedure, and add this flowchart segment to the growing object program solution flowchart. We first check the BTYPE field of the BLOCK structure found above. This is the field used to record the act of generating a block flowchart structure. It the field is blank, we set the field equal to 'DONE' and proceed to allocate a SELMNT flowchart structure. Then we fill in the relevant information, e.g. maximum iterations, block name, relative error, etc., from the appropriate fields in the BLOCK structure.

When we do allocate a block's flowchart structure, it is also necessary to compose for the code generation phase a list of FLD type nodes in our MSCC, so that these variables can be initialized prior to entry of the iterative solution procedure at runtime.

Before exiting from SIMUL_BLK, we also free up unneeded memory that was used for the SIM list and the node_id list. This is simple "house cleaning".

Finally, we pass the modified array graph to an offspring SCHEDULE_GRAPH for further scheduling. SCHEDULE_GRAPH will eventually return to us a pointer to a completed flowchart for the given subgraph which we in turn return to our calling program. In the event we added a block flowchart structure for a new block, we set its next-pointer equal to SCHEDULE_GRAPH's returned flowchart pointer and return a pointer to our generated flowchart.

These nine tasks comprise the processing activities of SIMUL_BLK. The above sequence of operations is performed by successive instances of SIMUL_BLK whenever SCHEDULE_GRAPH or SCHEDULE_COMPONENT confront an MSCC, until at last the entire object program flowchart has been generated.

5.3 EXAMPLES OF SCHEDULING SIMULTANEOUS EQUATIONS

Now we will examine a few examples of simple input specifications and their resultant object program flowcharts, in order to illustrate the scheduling of simultaneous blocks of equations in the MODEL system. The example specifications will be simple in order to illustrate certain specific points about the scheduling process.

Program flowcharts are described in associated flowchart reports, which are succinct representations of the object program's processing flow generated automatically for the user as part of MODEL's automatic documentation. Every time a user submits a specification to the MODEL system, several useful reports are generated, one of which is the Flowchart Report.

The flowchart report is a linearized list of actions which the object program has been designed to perform. Each line of this report identifies an action or a datum manipulated in the course of execution of the object program. Four columns are included in each line of the flowchart report: NODE$, NAME, DESCRIPTION and EVENT. The NODE$ is an integer which gives the internal system

identification number of a given graph node where appropriate. This number, which is useful in conjunction with some of the other MODEL output reports and to MODEL system programmers, can be disregarded in the following discussion. The second field is NAME, which identifies the assertion or qualified variable name manipulated in the current action. Recall that the name of an assertion is given by AASSid#, where id# is equal to ten times the line number on which the assertion appears plus its indentation number. The third field in the flowchart report is DESCRIPTION, which describes the entity given in the NAME column. The fourth and final column, EVENT, specifies what action is to be taken by the object program is manipulation of the associated program entity. In some cases, as for example when purely procedural events occur, no corresponding NAME exists in the input specification. The beginning and end of system added iterations are such events, so their NODE$ and NAME columns are blank in the flowchart report. For emphasis in presenting the use of iterative solution procedures and nesting, we have added brackets by hand enclosing the scope of such iterations in the following figures.

Figure 5-7 contains the simplest of our examples, a two equation simultaneous system within a single block. As can be easily seen in the accompanying flowchart report, the two equations are scheduled in the iterative solution procedure in the order that they appear in the input specification. Note that we have included a qualifier specifying the

maximum number of desired iterations (50) and that the system defaults for other values (relative error 0.0001, solution method Gauss-Seidel) have been relied upon.

In the next example, shown in Figure 5-8, we have retained the simple mathematical model of example 1, and added for illustration explicit qualifiers for relative error (one percent or 0.01) and solution method (Gauss-Seidel). See that the addition of these qualifiers does not alter the scheduled order of calculation as reflected in the accompanying flowchart. We have also added explicit initial values for X1 and X2 in this example, to be used by the code generation phase to assign initial values to the simultaneous variable before the Gauss-Seidel iterative procedure is begun.

In the next example, in Figure 5-9, we show how object program variable calculation order depends not only on the order of equations in the input specification, but also on the nature of data dependencies in the mathematical model. Here, there are four equations instead of two in the simultaneous block and it is up to the scheduler to determine which belong in the block, and in what order to calculate them. X1 and X2 depend on one another's values as before. X3 depends on the values of X1, X2 and X4, while X4 depends on none of the other variables, being equal to the constant value seventy-seven. At the beginning of scheduling, X4 can be scheduled right away therefore, leaving X1, X2 and X3 in the array graph. Furthermore, it

```
MODULE: EXAMPLE;
TARGET: EXAMOUT;

EXAMOUT IS FILE (OUTREC);
  OUTREC IS RECORD(X1,X2);
    (X1,X2) ARE FIELD (PIC'BBS99.9999');

BLOCK TRY: MAX ITER IS 50;
  X1 = 1 + 2 * X2;
  X2 = 1 - X1/4;
END TRY;
```

FLOWCHART REPORT

NODE$	NAME	DESCRIPTION	EVENT
37	EXAMPLE	MODULE NAME	PROCEDURE HEADING
		ITERATION(GAUSS_SEIDEL)	SIMULTANEOUS BLOCK: TRY
32	AASS90	ASSERTION	
35	EXAMOUT.X1	FIELD IN RECORD EXAMOUT.OUTREC	TARGET OF ASSERTION: AASS90
31	AASS100	ASSERTION	
36	EXAMOUT.X2	FIELD IN RECORD EXAMOUT.OUTREC	TARGET OF ASSERTION: AASS100
		END ITERATION	SIMULTANEOUS BLOCK: TRY
34	EXAMOUT.OUTREC	RECORD IN FILE EXAMOUT	WRITE RECORD
33	EXAMOUT	FILE	CLOSE FILE
		END	

Figure 5-7. MODEL Specification and Flowchart of a Simple System of Simultaneous Equations.

```
MODULE: EXAMPLE;
TARGET: EXAMOUT;

EXAMOUT IS FILE (OUTREC);
  OUTREC IS RECORD(X1,X2);
    (X1,X2) ARE FIELD (PIC'BBS99.9999');

INITIAL.X1 = 3**0.5;
INITIAL.X2 = 1400;

BLOCK TRY: MAX ITER IS 50,
           RELATIVE ERR IS 0.01,
           SOLUTION METHOD IS GAUSS_SEIDEL;
  X1 = 1 + 2 * X2;
  X2 = 1 - X1/4;
END TRY;
```

```
                                FLOWCHART REPORT
NODE$       NAME                DESCRIPTION                        EVENT
39    EXAMPLE                   MODULE NAME                        PROCEDURE HEADING
34    AASS90                    ASSERTION
40    INITIAL.EXAMOUT.X2        SPECIAL NAME                       TARGET OF ASSERTION: AASS90
33    AASS80                    ASSERTION
41    INITIAL.EXAMOUT.X1        SPECIAL NAME                       TARGET OF ASSERTION: AASS80
                                ITERATION(GAUSS_SEIDEL)            SIMULTANEOUS BLOCK: TRY
31    AASS140                   ASSERTION
37    EXAMOUT.X1                FIELD IN RECORD EXAMOUT.OUTREC     TARGET OF ASSERTION: AASS140
32    AASS150                   ASSERTION
38    EXAMOUT.X2                FIELD IN RECORD EXAMOUT.OUTREC     TARGET OF ASSERTION: AASS150
                                END ITERATION                      SIMULTANEOUS BLOCK: TRY
36    EXAMOUT.OUTREC            RECORD IN FILE EXAMOUT             WRITE RECORD
35    EXAMOUT                   FILE                               CLOSE FILE
                                END
```

Figure 5-8. MODEL Specification and Flowchart of a Simple System of Simultaneous Equations With Explicit Block Descriptors and Initial Values.

is found that only X1 and X2 are interdependent and must be solved together iteratively. Finally, X3 is calculated after the Gauss-Seidel procedure, when all its source variables have been found. Within the Gauss-Seidel iteration, calculation order corresponds to the order in the input specification. Scheduling of the variables X3 and X4 occur on the basis of data dependencies. They were removed from the scope of the iteration, increasing the efficiency of the computation without sacrificing correctness. In a simple model such as this, it would have been easy for the user to detect the nature of the data interrelationships and remove the assertions for X3 and X4 from the BLOCK himself. But in large and complicated systems of equations, such situations are difficult or impossible for humans to detect, making the job perfect for automated tools such as the present MODEL system.

In the simple case shown here, where no nesting of blocks occurs, removal of an assertion from the innermost block places it outside of the scope of any Gauss-Seidel iterations. When nested blocks and complex models are considered, dependencies might not dictate the removal of an assertion completely from the scope of such iterations, but might only indicate the removal to an outer level of nesting. This would still increase the efficiency of the resultant solution program, by decreasing the number of unnecessary recalculations of such variables.

The next two examples illustrate the use of nesting in

```
MODULE: EXAMPLE;
TARGET: EXAMOUT;

EXAMOUT IS FILE (OUTREC);
  OUTREC IS RECORD(X1,X2,X3,X4);
    (X1,X2,X3,X4) ARE FIELD (PIC'BBS99.9999');

BLOCK TRY: MAX ITER IS 50;
  X3 = X1 + X2 + X4;
  X1 = 1 + 2 * X2;
  X2 = 1 - X1/4;
  X4 = 77;
END TRY;
```

FLOWCHART REPORT

NODES	NAME	DESCRIPTION	EVENT
41	EXAMPLE	MODULE NAME	PROCEDURE HEADING
33	AASS120	ASSERTION	
40	EXAMOUT.X4	FIELD IN RECORD EXAMOUT.OUTREC	TARGET OF ASSERTION: AASS120
		ITERATION(GAUSS_SEIDEL)	SIMULTANEOUS BLOCK: TRY
31	AASS100	ASSERTION	
37	EXAMOUT.X1	FIELD IN RECORD EXAMOUT.OUTREC	TARGET OF ASSERTION: AASS100
32	AASS110	ASSERTION	
38	EXAMOUT.X2	FIELD IN RECORD EXAMOUT.OUTREC	TARGET OF ASSERTION: AASS110
		END ITERATION	SIMULTANEOUS BLOCK: TRY
34	AASS90	ASSERTION	
39	EXAMOUT.X3	FIELD IN RECORD EXAMOUT.OUTREC	TARGET OF ASSERTION: AASS90
36	EXAMOUT.OUTREC	RECORD IN FILE EXAMOUT	WRITE RECORD
35	EXAMOUT	FILE	CLOSE FILE
		END	

Figure 5-9. MODEL Specification and Flowchart of a Set of Simultaneous Equations Containing Simplifying Data Dependencies.

modelling with simultaneous equations. The first example, in Figure 5-10, is called NESTMOD and illustrates a nesting of two smaller blocks within one larger block. Here, all of the assertions are actually a part of the MSCC, as guaranteed by our somewhat artificial addition of the dependencies "+X-X" and "+Y-Y" in the defining assertions for variables A and C. The scheduler is therefore unable to dissociate any variables from the MSCC on the basis of missing dependencies as above, and so the flowchart order corresponds exactly to the order suggested in the input specification. The nesting of Gauss-Seidel procedures in the flowchart imitates the nesting of blocks in the input specification, and the dependencies do not give rise to any rearrangement of assertions.

In the last example, in Figure 5-11, we have removed the above mentionned artificial dependencies in the assertions defining A and C. Now it is found that, although the user requested a nested solution, one is not indicated on the basis of the data dependencies. In fact, blocks BLK2 and BLK3 are completely independent of one another and so can be solved in separate Gauss-Seidel iterations, with no need for nesting.

These two nesting examples illustrate how small changes in the dependencies of models can lead to rather large changes in the optimally efficient solution programs. It is clear that for all but the most trivial simultaneous models, a human cannot hope to produce such efficient solution

```
MODULE: NESTMOD;
SOURCE: NESTIN;
TARGET: NESTOUT;

NESTIN IS FILE (NESTREC);
  NESTREC IS RECORD(K(4));
    K IS FIELD (PIC'B9.V99');

NESTOUT IS FILE (OUTREC);
  OUTREC IS RECORD(X,Y,A,B,C,D);
    (X,Y,A,B,C,D) ARE FIELD (PIC'BBS(5)9.V(5)9');

BLOCK BLK1: MAX ITER IS 100;
  X = A * Y + B;
  BLOCK BLK2: MAX ITER IS 100;
    A = 0.2 * B + K(1) + X - X;
    B = 0.2 * A + K(2);
  END BLK2;
  Y = C * X + D;
  BLOCK BLK3: MAX ITER IS 100;
    C = 0.2 * D + K(3) + Y - Y;
    D = 0.2 * C + K(4);
  END BLK3;
END BLK1;
```

FLOWCHART REPORT

NODE$	NAME	DESCRIPTION	EVENT
40	NESTMOD	MODULE NAME	PROCEDURE HEADING
37	NESTIN	FILE	OPEN FILE
38	NESTIN.NESTREC	RECORD IN FILE NESTIN	READ RECORD
		ITERATION	FOR FOR_EACH.NESTIN.K UNTIL CONSTANT LIMITS
39	NESTIN.K	FIELD IN RECORD NESTIN.NESTREC	
		END ITERATION	FOR FOR_EACH.NESTIN.K
		ITERATION(GAUSS_SEIDEL)	SIMULTANEOUS BLOCK: BLK1
31	AASS140	ASSERTION	
43	NESTOUT.X	FIELD IN RECORD NESTOUT.OUTREC	TARGET OF ASSERTION: AASS140
		ITERATION(GAUSS_SEIDEL)	SIMULTANEOUS BLOCK: BLK2
32	AASS160	ASSERTION	
45	NESTOUT.A	FIELD IN RECORD NESTOUT.OUTREC	TARGET OF ASSERTION: AASS160
33	AASS170	ASSERTION	
46	NESTOUT.B	FIELD IN RECORD NESTOUT.OUTREC	TARGET OF ASSERTION: AASS170
		END ITERATION	SIMULTANEOUS BLOCK: BLK2
34	AASS190	ASSERTION	
44	NESTOUT.Y	FIELD IN RECORD NESTOUT.OUTREC	TARGET OF ASSERTION: AASS190
		ITERATION(GAUSS_SEIDEL)	SIMULTANEOUS BLOCK: BLK3
35	AASS210	ASSERTION	
47	NESTOUT.C	FIELD IN RECORD NESTOUT.OUTREC	TARGET OF ASSERTION: AASS210
36	AASS220	ASSERTION	
48	NESTOUT.D	FIELD IN RECORD NESTOUT.OUTREC	TARGET OF ASSERTION: AASS220
		END ITERATION	SIMULTANEOUS BLOCK: BLK3
		END ITERATION	SIMULTANEOUS BLOCK: BLK1
42	NESTOUT.OUTREC	RECORD IN FILE NESTOUT	WRITE RECORD
41	NESTOUT	FILE	CLOSE FILE
		END	

Figure 5-10. MODEL Specification and Flowchart of Nested Blocks of Simultaneous Equations.

```
MODULE: NESTMOD;
SOURCE: NESTIN;
TARGET: NESTOUT;

NESTIN IS FILE (NESTREC);
  NESTREC IS RECORD(K(4));
    K IS FIELD (PIC'B9.V99');

NESTOUT IS FILE (OUTREC);
  OUTREC IS RECORD(X,Y,A,B,C,D);
    (X,Y,A,B,C,D) ARE FIELD (PIC'BBS(5)9.V(5)9');

BLOCK BLK1: MAX ITER IS 100;
  X = A * Y + B;
  BLOCK BLK2: MAX ITER IS 100;
    A = 0.2 * B + K(1);
    B = 0.2 * A + K(2);
  END BLK2;
  Y = C * X + D;
  BLOCK BLK3: MAX ITER IS 100;
    C = 0.2 * D + K(3);
    D = 0.2 * C + K(4);
  END BLK3;
END BLK1;
```

FLOWCHART REPORT

NODE#	NAME	DESCRIPTION	EVENT
40	NESTMOD	MODULE NAME	PROCEDURE HEADING
37	NESTIN	FILE	OPEN FILE
38	NESTIN.NESTREC	RECORD IN FILE NESTIN	READ RECORD
		ITERATION	FOR FOR_EACH.NESTIN.K UNTIL CONSTANT LIMITS
39	NESTIN.K	FIELD IN RECORD NESTIN.NESTREC	
		END ITERATION	FOR FOR_EACH.NESTIN.K
		ITERATION(GAUSS_SEIDEL)	SIMULTANEOUS BLOCK: BLK3
35	AASS210	ASSERTION	
47	NESTOUT.C	FIELD IN RECORD NESTOUT.OUTREC	TARGET OF ASSERTION: AASS210
36	AASS220	ASSERTION	
48	NESTOUT.D	FIELD IN RECORD NESTOUT.OUTREC	TARGET OF ASSERTION: AASS220
		END ITERATION	SIMULTANEOUS BLOCK: BLK3
		ITERATION(GAUSS_SEIDEL)	SIMULTANEOUS BLOCK: BLK2
32	AASS160	ASSERTION	
45	NESTOUT.A	FIELD IN RECORD NESTOUT.OUTREC	TARGET OF ASSERTION: AASS160
33	AASS170	ASSERTION	
46	NESTOUT.B	FIELD IN RECORD NESTOUT.OUTREC	TARGET OF ASSERTION: AASS170
		END ITERATION	SIMULTANEOUS BLOCK: BLK2
		ITERATION(GAUSS_SEIDEL)	SIMULTANEOUS BLOCK: BLK1
31	AASS140	ASSERTION	
43	NESTOUT.X	FIELD IN RECORD NESTOUT.OUTREC	TARGET OF ASSERTION: AASS140
34	AASS190	ASSERTION	
44	NESTOUT.Y	FIELD IN RECORD NESTOUT.OUTREC	TARGET OF ASSERTION: AASS190
		END ITERATION	SIMULTANEOUS BLOCK: BLK1
42	NESTOUT.OUTREC	RECORD IN FILE NESTOUT	WRITE RECORD
41	NESTOUT	FILE	CLOSE FILE
		END	

Figure 5-11. MODEL Specification and Flowchart of Nested Blocks of Simultaneous Equations Containing Simplifying Data Dependencies.

programs without the use of an automated system such as MODEL.

In the next chapter of this report, we proceed to the code generation phase of the MODEL processor, to examine the process of creating from our flowchart a correct iterative solution procedure.

CHAPTER 6

SIMULTANEOUS EQUATIONS: CODE GENERATION

In this chapter we examine the transformation of our completed object program flowchart into a correct and functional PL/I program. Since the algorithms employed during code generation are relatively straightforward and much less novel than those used in the scheduling process, we will concentrate mainly on the structure of the produced PL/I programs, rather than on the method of production of these programs. Some examples of generated programs will be presented which correspond to the input specifications and flowchart reports of the previous chapter of this report.

6.1 THE ITERATIVE SOLUTION METHOD

The iterative solution method employed in code generation is the Gauss-Seidel method, a well known algorithm which iteratively recalculates variables until the difference between successive values of all variables do not exceed a specified threshold, at which time the system of equations is said to have converged. If, after a set number

of iterations, not all variables in the system meet this convergence criterion, then the set of equations is said to have diverged. Depending on the behavior of the set of equations, the output we would normally expect from such a procedure is either

a) the correct solution values of variables within the set of equations, or

b) the names of those variables which have not converged after some maximum number of iterations has been achieved.

In the following discussion, we will see how MODEL generated programs efficiently produce exactly these desired results.

For each block of simultaneous equations to be solved in the system flowchart, the MODEL processor generates three logical sections of PL/I source code:

1) an initialization sequence,

2) a convergence procedure, and

3) a recalculation loop.

6.1.1 THE INITIALIZATION SEQUENCE

The first step in the iterative solution of a system of simultaneous equations is to assign all initial values which serve as starting points in the recalculation process. Initial values for variables can be obtained from two sources: user specified assertions or system generated default values.

When the user has supplied an initial value via the INITIAL statement, the system will have generated statements which assign these values to special "INITIAL$" variables by the time the Gauss-Seidel source code is to be generated. In this case, all that remains is to assign these INITIAL$ variables to their appropriate solution variables. For example, suppose our user wished to assign initial values of five and seven to variables X and Y, respectively. He would signify this by including in his specification the assertions

```
INITIAL.X = 5;
INITIAL.Y = 7;
```

Before the Gauss-Seidel source code is generated, the system will have generated code assigning these values to special INITIAL$ variables, as in the PL/I code:

```
INITIAL$X = 5;
INITIAL$Y = 7;
```

All that now remains is to assign these INITIAL$ variables to their appropriate solution variables, as in the PL/I code:

```
X = INITIAL$X;
Y = INITIAL$Y;
```

to ensure that the variables X and Y have been properly initialized.

If the user omits statements specifying initial values from his specification, then the system generates initial values for all necessary variables according to the policy presented in Chapter five of this report:

- if the variable is a zero dimensional entity, i.e. a simple variable, then it is assigned the initial value one.

- if the variable is of dimension one or greater, i.e. a vector or a multidimensional array, then the first data element along each dimension is assigned the initial value one; subsequent elements along any given dimension are assigned initial values equal to the solution values of their immediate predecessors along that dimension.

The latter policy element is possible because MODEL uniformly generates solutions for muldimensional variables in order of increasing range indices. That is, subscript values begin with the index value one and increase monotonically up to the SIZE of the given dimension.

Thus, if X were a variable of zero dimension, e.g. a simple integer variable, the system generates PL/I code of

```
X = 1;
```

If X is a one dimsnsional array (a vector), then the PL/I code generated is

```
IF $I=1 THEN X($I)=1;
ELSE X($I)=X($I-1);
```

where $I is a subscript ranging along the length of the vector X. In the latter case, the solution takes place within an iteration on $I so that each previous variable has been solved by the time any given element after the first is initialized. The above example can easily be generalized to handle data structures of arbitrary range and

dimensionality.

6.1.2 THE CONVERGENCE PROCEDURE

The convergence procedure, as we call it, is a local PL/I procedure generated for each block of simultaneous equations. It performs the function of convergence testing and error message creation for the corresponding block's recalculation loop described later. At each new recalculation of a variable, this convergence procedure is called with three arguments: the newly recalculated value of the variable, its previous value and the documentation name of the variable. If the new value is within the convergence criterion distance of the previous value, then no action is taken. But if it is not, then a flag is cleared to indicate this lack of convergence within the corresponding block, indicating the need for further iterations. Furthermore, if convergence as indicated by these values is not obtained and the maximum number of prescribed iterations has been attained, then an error message is created and sent to the user informing him which block and which variables have not converged.

The name of each convergence procedure is chosen according to the documentation name of the corresponding block given by the user in his input specification. If the block name given in the BLOCK statement is XYZ, for example, then the convergence procedure name would be $ITER_PROC_XYZ in the generated PL/I program. For each block of equations,

the name of the convergence procedure is formed by prefixing "$ITER_PROC_" to the BLOCK name supplied by the user. Use of this name in generating source code helps make the resultant PL/I program easier to read, should the user wish to do so.

The three variables passed to each convergence procedure are known locally in each procedure by the names $OLD_VAL, $NEW_VAL and $VAR_NAME. The titles of these variables are designed to be descriptive of their functions. $NEW_VAL and $OLD_VAL, both floating point numbers, hold the newly recalculated and previous values, respectively, of the particular variable being currently tested for convergence. $VAR_NAME, a character string variable, holds the name of the variable currently being tested, and is used to construct error messages in the event of divergence.

When a convergence procedure is called, it first tests the new and previous values of the variable against the relative convergence criterion which was obtained by the code generator from the block flowchart data structure created during the scheduling phase. This information, along with the block name and maximum number of iterations, is built into the convergence procedure for each block being solved. An example of generated PL/I code to test for convergence is

```
IF $OLD_VAL = 0.0 THEN $REL_ERR = ABS($NEW_VAL);
ELSE $REL_ERR = ABS($OLD_VAL-$NEW_VAL)/ABS($OLD_VAL);
IF $REL_ERR > 1.0E-04 THEN DO;
```

where ABS is the absolute value function. This statement tests the current variable for convergence against the relative convergence criterion 0.0001, or 1/100 percent. The floating point variable $REL_ERR, is used to hold the value of the relative error or distance between the old and new values of the variable being tested. Note that, if the previous value of the variable is zero, then the relative distance cannot be calculated algegraically since this would lead to division by zero. In this case, therefore, we take the value of the relative error to be the new value of the variable being tested. If the above test, which is actually passed if divergence is indicated (i.e. the test is for >), shows that the variable does not meet the convergence criterion, then the convergence procedure goes on to test whether the maximum number of iterations have been performed. If so, then error messages containing the block name, the name of the divergent variable, its current and previous values and the relative distance between them are prepared in an error message buffer ($ERROR_BUF) and output to a system error message file (ERRORF). Whenever a variable does not pass the convergence test, a flag is cleared to indicate that additional iterations are required. Thus, a complete convergence procedure for a block named TRY with relative convergence criterion of 1/100 percent and maximum number of iterations of fifty would be:

```
$ITER_PROC_TRY: PROCEDURE($OLD_VAL,$NEW_VAL,$VAR_NAME);
  DCL ($OLD_VAL,$NEW_VAL,$REL_ERR) FLOAT BIN;
  DCL ($VAR_NAME) CHAR(32) VAR;
  IF $OLD_VAL = 0.0 THEN $REL_ERR = ABS($NEW_VAL);
  ELSE $REL_ERR = ABS($OLD_VAL-$NEW_VAL)/ABS($OLD_VAL);
  IF $REL_ERR > 1.0E-04 THEN DO;
    IF $ITER_CNTR_1 = 50 THEN DO;
      $ERROR_BUF = 'NO CONVERGENCE IN BLOCK:  TRY';
      WRITE FILE (ERRORF) FROM ($ERROR_BUF);
      $ERROR_BUF = ' VARIABLE NAME :  '|| $VAR_NAME;
      WRITE FILE (ERRORF) FROM ($ERROR_BUF);
      $ERROR_BUF = ' CURRENT VALUE :  '|| $NEW_VAL;
      WRITE FILE (ERRORF) FROM ($ERROR_BUF);
      $ERROR_BUF = ' PREVIOUS VALUE:  '|| $OLD_VAL;
      WRITE FILE (ERRORF) FROM ($ERROR_BUF);
      $ERROR_BUF = ' RELATIVE ERROR:  '|| $REL_ERR;
      WRITE FILE (ERRORF) FROM ($ERROR_BUF);
    END;
    $ITER_CNVRG_1 = '0'B;
  END;
  RETURN;
END $ITER_PROC_TRY;
```

Here $ITER_CNTR_1 is a counter of Gauss-Seidel iterations and $ITER_CNVRG_1 is the flag used to indicate failure of the convergence test. The derivation of the names for these variables is presented in the following discussion.

6.1.3 THE GAUSS-SEIDEL RECALCULATION LOOP

The actual recalculation of variables which is at the heart of the Gauss-Seidel algorithm is performed in these iterative recalculation loops. Each loop, one for each block in the solution program, calculates new values for the system of equation's variables in the order prescribed by the generated flowchart and calls the correct convergence procedure at each recalculation to test for convergence. In each pass through an iterative loop, every variable in that loop is recalculated once and tested. The iterations continue until every variable in the loop has passed the

convergence test in the current iteration, or until the maximum number of iterations has been reached and not all variables have passed the convergence test, whichever comes first. Then the system of equations will have converged or diverged, respectively.

A recalculation and test of a single variable consists of saving the old value, calculating the new value according to the user's assertion, and calling the corresponding convergence procedure for that variable. For example, if a user's assertion in BLOCK TRY is

```
a: X1 = 1 + 2*X2;
```

then the corresponding recalculation and test in the generated PL/I source code would be

```
$TMP_VAL = X1;
X1 = 1 + 2*X2;
CALL $ITER_PROC_TRY($TMP_VAL,X1,'X1');
```

where $TMP_VAL is a global variable used to hold temporary previous values of variables while they are recalculated. (Note that, for simplicity, we have not qualified the variable names in the PL/I code shown here with their data file names. In actual generated code these qualifiers do appear. For example, if X1 and X2 belonged to file OUT, then the actual source code produced would be

```
OUT.X1 = 1 + 2*OUT.X2;
```

for this equation. For clarity we omit this detail in the current examples.)

Every variable is recalculated in a given iteration.

The iterations continue until either convergence is attained or the maximum number of iterations is reached, whichever comes first. Thus, if our iteration counter is called $ITER_CNTR_1 and our convergence flag is called $ITER_CNVRG_1, then the Gauss-Seidel iteration would be:

```
$ITER_CNVRG_1 = '0'B;
DO $ITER_CNTR_1 = 1 TO 50 WHILE (^$ITER_CNVRG_1);
     $ITER_CNVRG_1 = '1'B;
     <list of variable recalculations>
END;
```

in the generated PL/I program, where the maximum number of iterations is taken to be fifty. At the beginning of each iteration, the flag $ITER_CNVRG_1 is set to one, a value indicating convergence. This amounts to an implicit assumption of convergence at the top of each iteration. If, during the convergence tests in the body of the loop, any variable is found not to pass its convergence test, then $ITER_CNVRG_1 will be cleared (i.e. set equal to the Boolean zero or FALSE value) by the loop's convergence procedure. Iterations continue until the maximum number of iterations have been reached, fifty in this case, or until $ITER_CNVRG_1 has the value one (i.e. TRUE) at the top of the loop, indicating that all the variables in the loop have passed the convergence test.

The names of the variables used to count iterations and to test for convergence are formed by prepending the strings "$ITER_CNTR_" and "$ITER_CNVRG_", respectively, onto an integer giving the nesting level of the corresponding block

of equations. Thus, when several blocks exist at the same level of nesting, they all share a common convergence testing flag and iteration counter. This aids readability of the generated PL/I program when complicated nesting relationships exist within a solution program. In the example above, this integer has the value one, indicating that the block of equations we are solving is at the first level of nesting.

When nesting does exist, an additional PL/I statement must be added to each Gauss-Seidel loop after the closure of the loop, to ensure strict convergence of the entire system. This is because an entire nested structure is deemed to converge only when every variable in every nesting level of the entire system has converged in the same iteration of the outermost level. Thus, at the end of an iteration, not only must the outermost convergence flag be equal to one to indicate convergence of the system, but every internal Gauss-Seidel loop must have a value of one in its local iteration counter after passing its own internal convergence tests. Only then will every variable in the system have passed its convergence test in the latest pass of every iteration loop. To enforce this additional criterion when nested blocks of equations are present, we add the following PL/I statement immediately after a Gauss-Seidel loop on nesting level n of our system:

```
IF $ITER_CNTR_n ^= 1 THEN $ITER_CNTR_(n-1) = '0'B;
```

That is, if any internal block fails to converge in a single iteration, we prevent its containing block from converging

by clearing its convergence test flag.

6.2 EXAMPLES OF ITERATIVE SOLUTION PROGRAMS

Below we present the object PL/I source code which corresponds to the input specifications and their flowcharts illustrated in the previous chapter of this report. Figures 6-1 through 6-5 here correspond exactly with Figures 5-7 through 5-11 in that chapter. For brevity, we only show those portions of the PL/I programs which pertain directly to the solution of simultaneous equations, omitting those sections of source code dealing with data declarations, input/output, etc. Only Figure 6-1 shows the entire object program, included for completeness. In these examples, recall the following illustrative characteristics:

Figure 6-1 - a simple 2-equation model with default block characteristics.

Figure 6-2 - the same model with user-specified initial values, convergence criterion, and maximum number of iterations.

Figure 6-3 - a 4-equation model, only two equations found to be in the MSCC.

Figure 6-4 - a set of three nested blocks which was unravelled into three consecutive blocks.

Figure 6-5 - a correctly nested system of three blocks and two nesting levels.

In the following chapter of this report we present a major application of the new MODEL facilities which handle

```
EXAMPLE: PROCEDURE OPTIONS(MAIN);
DCL $ITER_CNVRG_1 BIT(1) ;
DCL $ITER_CNTR_1 FIXED BIN ;
DCL EXAMOUT_OUTREC_PTR PTR;
DCL EXAMOUT_OUTREC_S CHAR(20) VARYING INIT('');
DCL EXAMOUT_OUTREC_P CHAR(20) BASED (EXAMOUT_OUTREC_PTR);
DCL EXAMOUTT RECORD SEQL OUTPUT;
DCL $FSTEXAMOUTT BIT(1) INIT('1'B);
DCL $ERROR_BUF CHAR(270) VAR;
DCL ERRORF FILE RECORD OUTPUT;
DCL ERRORF_BIT BIT(1) STATIC INIT('1'B);
DCL ($ERROR,$ACC_ERROR,$NOT_DONE)(20) BIT(1);
DCL $ERR_LAB(20) LABEL;
DCL $ERRSP$ FIXED BIN STATIC INITIAL (0);
DCL $TMP_VAL FLOAT BIN;
DCL $TMP_ERR BIT(1);
DECLARE
   1 EXAMOUT,
    2 OUTREC,
     3 X1 PIC'BBS99.9999',
     3 X2 PIC'BBS99.9999';
DCL (TRUE,SELECTED) BIT(1) INIT('1'B);
DCL (FALSE,NOT_SELE,NOT_SELECTED) BIT(1) INIT('0'B);
ON UNDEFINEDFILE(ERRORF)  ERRORF_BIT='0'B;
ERROR_RESTART: ;
$ERRSP$ = $ERRSP$ +1;
$ERROR($ERRSP$)='0'B;
$ACC_ERROR($ERRSP$)='0'B;
$ERR_LAB($ERRSP$)=END_PROGRAM;
EXAMOUT.X2 = 1;                                          ] initialization sequence
EXAMOUT.X1 = 1;                                          ]
$ITER_PROC_TRY: PROCEDURE($OLD_VAL,$NEW_VAL,$VAR_NAME);  ]
  DCL ($OLD_VAL,$NEW_VAL,$REL_ERR) FLOAT BIN;            ]
  DCL $VAR_NAME CHAR(32) VAR;                            ]
  IF $OLD_VAL = 0.0 THEN $REL_ERR = ABS($NEW_VAL);       ]
  ELSE $REL_ERR = ABS($OLD_VAL-$NEW_VAL)/ABS($OLD_VAL);  ]
  IF $REL_ERR > 1.000000E-04 THEN DO;                    ]
    IF $ITER_CNTR_1 = 50 THEN DO;                        ]
      $ERROR_BUF = 'NO CONVERGENCE IN BLOCK: TRY';       ]
      WRITE FILE (ERRORF) FROM ($ERROR_BUF);             ]
      $ERROR_BUF = '  VARIABLE NAME :  '|| $VAR_NAME;    ]
      WRITE FILE (ERRORF) FROM ($ERROR_BUF);             ]
      $ERROR_BUF = '  CURRENT VALUE : '|| $NEW_VAL;      ]
      WRITE FILE (ERRORF) FROM ($ERROR_BUF);             ]
      $ERROR_BUF = '  PREVIOUS VALUE: '|| $OLD_VAL;      ] convergence procedure
      WRITE FILE (ERRORF) FROM ($ERROR_BUF);             ]
      $ERROR_BUF = '  RELATIVE ERROR: '|| $REL_ERR;      ]
      WRITE FILE (ERRORF) FROM ($ERROR_BUF);             ]
    END;                                                 ]
    $ITER_CNVRG_1 = '0'B;                                ]
  END;                                                   ]
  RETURN;                                                ]
END $ITER_PROC_TRY;                                      ]
$ITER_CNVRG_1 = '0'B;                                    ]
DO $ITER_CNTR_1 = 1 TO 50 WHILE (^$ITER_CNVRG_1);        ]
  $ITER_CNVRG_1= '1'B;                                   ]
  $TMP_VAL = EXAMOUT.X1;                                 ]
  EXAMOUT.X1 = 1+2*EXAMOUT.X2;                           ] recalculation loop
  CALL $ITER_PROC_TRY($TMP_VAL,EXAMOUT.X1,'EXAMOUT.X1'); ]
  $TMP_VAL = EXAMOUT.X2;                                 ]
  EXAMOUT.X2 = 1-EXAMOUT.X1/4;                           ]
  CALL $ITER_PROC_TRY($TMP_VAL,EXAMOUT.X2,'EXAMOUT.X2'); ]
END /* $ITER_CNTR_1 */ ;                                ]
EXAMOUT_OUTREC_PTR=ADDR(EXAMOUT.OUTREC);
EXAMOUT_OUTREC_S = EXAMOUT_OUTREC_P;
WRITE FILE(EXAMOUTT) FROM (EXAMOUT_OUTREC_S);
EXAMOUT_OUTREC_S='';
CLOSE FILE(EXAMOUTT);
END_PROGRAM: RETURN;
END EXAMPLE;
```

Figure 6-1. PL/I Solution Program Corresponding to Example in Figure 5-7.

```
INITIAL$EXAMOUT_X2=1400;
INITIAL$EXAMOUT_X1=3**0.5;
EXAMOUT.X2 = INITIAL$EXAMOUT_X2;
EXAMOUT.X1 = INITIAL$EXAMOUT_X1;
$ITER_PROC_TRY: PROCEDURE($OLD_VAL,$NEW_VAL,$VAR_NAME);
  DCL ($OLD_VAL,$NEW_VAL,$REL_ERR) FLOAT BIN;
  DCL $VAR_NAME CHAR(32) VAR;
  IF $OLD_VAL = 0.0 THEN $REL_ERR = ABS($NEW_VAL);
  ELSE $REL_ERR = ABS($OLD_VAL-$NEW_VAL)/ABS($OLD_VAL);
  IF $REL_ERR >  1.000000E-02 THEN DO;
    IF $ITER_CNTR_1 = 50 THEN DO;
      $ERROR_BUF = 'NO CONVERGENCE IN BLOCK: TRY';
      WRITE FILE (ERRORF) FROM ($ERROR_BUF);
      $ERROR_BUF = '  VARIABLE NAME :  '|| $VAR_NAME;
      WRITE FILE (ERRORF) FROM ($ERROR_BUF);
      $ERROR_BUF = '  CURRENT VALUE : '|| $NEW_VAL;
      WRITE FILE (ERRORF) FROM ($ERROR_BUF);
      $ERROR_BUF = '  PREVIOUS VALUE: '|| $OLD_VAL;
      WRITE FILE (ERRORF) FROM ($ERROR_BUF);
      $ERROR_BUF = '  RELATIVE ERROR: '|| $REL_ERR;
      WRITE FILE (ERRORF) FROM ($ERROR_BUF);
    END;
    $ITER_CNVRG_1 = '0'B;
  END;
  RETURN;
END $ITER_PROC_TRY;
$ITER_CNVRG_1 = '0'B;
DO $ITER_CNTR_1 = 1 TO 50 WHILE (^$ITER_CNVRG_1);
  $ITER_CNVRG_1= '1'B;
  $TMP_VAL = EXAMOUT.X1;
  EXAMOUT.X1 = 1+2*EXAMOUT.X2;
  CALL $ITER_PROC_TRY($TMP_VAL,EXAMOUT.X1,'EXAMOUT.X1');
  $TMP_VAL = EXAMOUT.X2;
  EXAMOUT.X2 = 1-EXAMOUT.X1/4;
  CALL $ITER_PROC_TRY($TMP_VAL,EXAMOUT.X2,'EXAMOUT.X2');
END /* $ITER_CNTR_1 */ ;
```

initialization sequence

convergence procedure

recalculation loop

Figure 6-2. Excerpt from PL/I Solution Program Corresponding to Example in Figure 5-8.

```
EXAMOUT.X4=77;
EXAMOUT.X2 = 1;
EXAMOUT.X1 = 1;
$ITER_PROC_TRY: PROCEDURE($OLD_VAL,$NEW_VAL,$VAR_NAME);
  DCL ($OLD_VAL,$NEW_VAL,$REL_ERR) FLOAT BIN;
  DCL $VAR_NAME CHAR(32) VAR;
  IF $OLD_VAL = 0.0 THEN $REL_ERR = ABS($NEW_VAL);
  ELSE $REL_ERR = ABS($OLD_VAL-$NEW_VAL)/ABS($OLD_VAL);
  IF $REL_ERR >  1.000000E-04 THEN DO;
    IF $ITER_CNTR_1 = 50 THEN DO;
      $ERROR_BUF = 'NO CONVERGENCE IN BLOCK: TRY';
      WRITE FILE (ERRORF) FROM ($ERROR_BUF);
      $ERROR_BUF = '  VARIABLE NAME :  '|| $VAR_NAME;
      WRITE FILE (ERRORF) FROM ($ERROR_BUF);
      $ERROR_BUF = '  CURRENT VALUE : '|| $NEW_VAL;
      WRITE FILE (ERRORF) FROM ($ERROR_BUF);
      $ERROR_BUF = '  PREVIOUS VALUE: '|| $OLD_VAL;
      WRITE FILE (ERRORF) FROM ($ERROR_BUF);
      $ERROR_BUF = '  RELATIVE ERROR: '|| $REL_ERR;
      WRITE FILE (ERRORF) FROM ($ERROR_BUF);
    END;
    $ITER_CNVRG_1 = '0'B;
  END;
  RETURN;
END $ITER_PROC_TRY;
$ITER_CNVRG_1 = '0'B;
DO $ITER_CNTR_1 = 1 TO 50 WHILE (^$ITER_CNVRG_1);
  $ITER_CNVRG_1= '1'B;
  $TMP_VAL = EXAMOUT.X1;
  EXAMOUT.X1 = 1+2*EXAMOUT.X2;
  CALL $ITER_PROC_TRY($TMP_VAL,EXAMOUT.X1,'EXAMOUT.X1');
  $TMP_VAL = EXAMOUT.X2;
  EXAMOUT.X2 = 1-EXAMOUT.X1/4;
  CALL $ITER_PROC_TRY($TMP_VAL,EXAMOUT.X2,'EXAMOUT.X2');
END /* $ITER_CNTR_1 */ ;
EXAMOUT.X3=EXAMOUT.X1+EXAMOUT.X2+EXAMOUT.X4;
```

initialization sequence

convergence procedure

recalculation loop

Figure 6-3. Excerpt from PL/I Solution Program Corresponding to Example in Figure 5-9.

```
NESTOUT.D = 1;
NESTOUT.C = 1;
NESTOUT.Y = 1;
NESTOUT.B = 1;
NESTOUT.A = 1;
NESTOUT.X = 1;
$ITER_PROC_BLK1: PROCEDURE($OLD_VAL,$NEW_VAL,$VAR_NAME);
  DCL ($OLD_VAL,$NEW_VAL,$REL_ERR) FLOAT BIN;
  DCL $VAR_NAME CHAR(32) VAR;
  IF $OLD_VAL = 0.0 THEN $REL_ERR = ABS($NEW_VAL);
  ELSE $REL_ERR = ABS($OLD_VAL-$NEW_VAL)/ABS($OLD_VAL);
  IF $REL_ERR > 1.000000E-04 THEN DO;
    IF $ITER_CNTR_1 = 100 THEN DO;
      [ print error messages indicating no convergence ]
    END;
    $ITER_CNVRG_1 = '0'B;
  END;
  RETURN;
END $ITER_PROC_BLK1;
$ITER_CNVRG_1 = '0'B;
DO $ITER_CNTR_1 = 1 TO 100 WHILE (^$ITER_CNVRG_1);
  $ITER_CNVRG_1= '1'B;
  $TMP_VAL = NESTOUT.X;
  NESTOUT.X = NESTOUT.A*NESTOUT.Y+NESTOUT.B;
  CALL $ITER_PROC_BLK1($TMP_VAL,NESTOUT.X,'NESTOUT.X');
  $ITER_PROC_BLK2: PROCEDURE($OLD_VAL,$NEW_VAL,$VAR_NAME);
    DCL ($OLD_VAL,$NEW_VAL,$REL_ERR) FLOAT BIN;
    DCL $VAR_NAME CHAR(32) VAR;
    IF $OLD_VAL = 0.0 THEN $REL_ERR = ABS($NEW_VAL);
    ELSE $REL_ERR = ABS($OLD_VAL-$NEW_VAL)/ABS($OLD_VAL);
    IF $REL_ERR > 1.000000E-04 THEN DO;
      IF $ITER_CNTR_2 = 100 THEN DO;
        [ print error messages indicating no convergence ]
      END;
      $ITER_CNVRG_2 = '0'B;
    END;
    RETURN;
  END $ITER_PROC_BLK2;
  $ITER_CNVRG_2 = '0'B;
  DO $ITER_CNTR_2 = 1 TO 100 WHILE (^$ITER_CNVRG_2);
    $ITER_CNVRG_2= '1'B;
    $TMP_VAL = NESTOUT.A;
    NESTOUT.A = 0.2*NESTOUT.B+NESTIN.K(1)+NESTOUT.X-NESTOUT.X;
    CALL $ITER_PROC_BLK2($TMP_VAL,NESTOUT.A,'NESTOUT.A');
    $TMP_VAL = NESTOUT.B;
    NESTOUT.B = 0.2*NESTOUT.A+NESTIN.K(2);
    CALL $ITER_PROC_BLK2($TMP_VAL,NESTOUT.B,'NESTOUT.B');
  END /* $ITER_CNTR_2 */ ;
  IF $ITER_CNTR_2 ^= 2 THEN $ITER_CNVRG_1 = '0'B;
  $TMP_VAL = NESTOUT.Y;
  NESTOUT.Y = NESTOUT.C*NESTOUT.X+NESTOUT.D;
  CALL $ITER_PROC_BLK1($TMP_VAL,NESTOUT.Y,'NESTOUT.Y');
  $ITER_PROC_BLK3: PROCEDURE($OLD_VAL,$NEW_VAL,$VAR_NAME);
    DCL ($OLD_VAL,$NEW_VAL,$REL_ERR) FLOAT BIN;
    DCL $VAR_NAME CHAR(32) VAR;
    IF $OLD_VAL = 0.0 THEN $REL_ERR = ABS($NEW_VAL);
    ELSE $REL_ERR = ABS($OLD_VAL-$NEW_VAL)/ABS($OLD_VAL);
    IF $REL_ERR > 1.000000E-04 THEN DO;
      IF $ITER_CNTR_2 = 100 THEN DO;
        [ print error messages indicating no convergence ]
      END;
      $ITER_CNVRG_2 = '0'B;
    END;
    RETURN;
  END $ITER_PROC_BLK3;
  $ITER_CNVRG_2 = '0'B;
  DO $ITER_CNTR_2 = 1 TO 100 WHILE (^$ITER_CNVRG_2);
    $ITER_CNVRG_2= '1'B;
    $TMP_VAL = NESTOUT.C;
    NESTOUT.C = 0.2*NESTOUT.D+NESTIN.K(3)+NESTOUT.Y-NESTOUT.Y;
    CALL $ITER_PROC_BLK3($TMP_VAL,NESTOUT.C,'NESTOUT.C');
    $TMP_VAL = NESTOUT.D;
    NESTOUT.D = 0.2*NESTOUT.C+NESTIN.K(4);
    CALL $ITER_PROC_BLK3($TMP_VAL,NESTOUT.D,'NESTOUT.D');
  END /* $ITER_CNTR_2 */ ;
  IF $ITER_CNTR_2 ^= 2 THEN $ITER_CNVRG_1 = '0'B;
END /* $ITER_CNTR_1 */ ;
```

Figure 6-4. Excerpt from PL/I Solution Program Corresponding to Example in Figure 5-10.

```
NESTOUT.D = 1;
NESTOUT.C = 1;
$ITER_PROC_BLK3: PROCEDURE($OLD_VAL,$NEW_VAL,$VAR_NAME);
  DCL ($OLD_VAL,$NEW_VAL,$REL_ERR) FLOAT BIN;
  DCL $VAR_NAME CHAR(32) VAR;
  IF $OLD_VAL = 0.0 THEN $REL_ERR = ABS($NEW_VAL);
  ELSE $REL_ERR = ABS($OLD_VAL-$NEW_VAL)/ABS($OLD_VAL);
  IF $REL_ERR >  1.000000E-04 THEN DO;
    IF $ITER_CNTR_1 = 100 THEN DO;
      [ print error messages indicating no convergence ]
    END;
    $ITER_CNVRG_1 = '0'B;
  END;
  RETURN;
END $ITER_PROC_BLK3;
$ITER_CNVRG_1 = '0'B;
DO $ITER_CNTR_1 = 1 TO 100 WHILE (^$ITER_CNVRG_1);
  $ITER_CNVRG_1= '1'B;
  $TMP_VAL = NESTOUT.C;
  NESTOUT.C = 0.2*NESTOUT.D+NESTIN.K(3);
  CALL $ITER_PROC_BLK3($TMP_VAL,NESTOUT.C,'NESTOUT.C');
  $TMP_VAL = NESTOUT.D;
  NESTOUT.D = 0.2*NESTOUT.C+NESTIN.K(4);
  CALL $ITER_PROC_BLK3($TMP_VAL,NESTOUT.D,'NESTOUT.D');
END /* $ITER_CNTR_1 */ ;
NESTOUT.B = 1;
NESTOUT.A = 1;
$ITER_PROC_BLK2: PROCEDURE($OLD_VAL,$NEW_VAL,$VAR_NAME);
  DCL ($OLD_VAL,$NEW_VAL,$REL_ERR) FLOAT BIN;
  DCL $VAR_NAME CHAR(32) VAR;
  IF $OLD_VAL = 0.0 THEN $REL_ERR = ABS($NEW_VAL);
  ELSE $REL_ERR = ABS($OLD_VAL-$NEW_VAL)/ABS($OLD_VAL);
  IF $REL_ERR >  1.000000E-04 THEN DO;
    IF $ITER_CNTR_1 = 100 THEN DO;
      [ print error messages indicating no convergence ]
    END;
    $ITER_CNVRG_1 = '0'B;
  END;
  RETURN;
END $ITER_PROC_BLK2;
$ITER_CNVRG_1 = '0'B;
DO $ITER_CNTR_1 = 1 TO 100 WHILE (^$ITER_CNVRG_1);
  $ITER_CNVRG_1= '1'B;
  $TMP_VAL = NESTOUT.A;
  NESTOUT.A = 0.2*NESTOUT.B+NESTIN.K(1);
  CALL $ITER_PROC_BLK2($TMP_VAL,NESTOUT.A,'NESTOUT.A');
  $TMP_VAL = NESTOUT.B;
  NESTOUT.B = 0.2*NESTOUT.A+NESTIN.K(2);
  CALL $ITER_PROC_BLK2($TMP_VAL,NESTOUT.B,'NESTOUT.B');
END /* $ITER_CNTR_1 */ ;
NESTOUT.Y = 1;
NESTOUT.X = 1;
$ITER_PROC_BLK1: PROCEDURE($OLD_VAL,$NEW_VAL,$VAR_NAME);
  DCL ($OLD_VAL,$NEW_VAL,$REL_ERR) FLOAT BIN;
  DCL $VAR_NAME CHAR(32) VAR;
  IF $OLD_VAL = 0.0 THEN $REL_ERR = ABS($NEW_VAL);
  ELSE $REL_ERR = ABS($OLD_VAL-$NEW_VAL)/ABS($OLD_VAL);
  IF $REL_ERR >  1.000000E-04 THEN DO;
    IF $ITER_CNTR_1 = 100 THEN DO;
      [ print error messages indicating no convergence ]
    END;
    $ITER_CNVRG_1 = '0'B;
  END;
  RETURN;
END $ITER_PROC_BLK1;
$ITER_CNVRG_1 = '0'B;
DO $ITER_CNTR_1 = 1 TO 100 WHILE (^$ITER_CNVRG_1);
  $ITER_CNVRG_1= '1'B;
  $TMP_VAL = NESTOUT.X;
  NESTOUT.X = NESTOUT.A*NESTOUT.Y+NESTOUT.B;
  CALL $ITER_PROC_BLK1($TMP_VAL,NESTOUT.X,'NESTOUT.X');
  $TMP_VAL = NESTOUT.Y;
  NESTOUT.Y = NESTOUT.C*NESTOUT.X+NESTOUT.D;
  CALL $ITER_PROC_BLK1($TMP_VAL,NESTOUT.Y,'NESTOUT.Y');
END /* $ITER_CNTR_1 */ ;
```

Figure 6-5. Excerpt from PL/I Solution Program Corresponding to Example in Figure 5-11.

systems of simultaneous equations. This application is a multimodel linked world trade model completely specified using the MODEL language.

CHAPTER 7

THE MODEL-LINK WORLD TRADE MODEL

In this chapter we describe the linked world trade model generated in the final stage of this research project using the tools described in the preceding chapters. A description of the model's organization, output reports, data files and linkage model are presented here. For a full listing of the model, its PL/I object program implementation and the automatically generated documentation produced by the MODEL system's compilation of this specification, see Appendix IV.

7.1 PHILOSOPHY AND ORGANIZATION OF THE MODEL

The purpose of this phase of research was to demonstrate the usefulness of the MODEL system in full-scale econometric modelling. To do so , we combined our two previously completed specifications of Spain and France with a simple model of the Rest of the World (ROW) and a novel linkage model into a single specification in the MODEL language. The PL/I object program which resulted from

compilation of this specification was used with data obtained from Project LINK's central data base to simulate the performance of the world economy from 1978 to 1982. Because the dynamic portion of our model was small, containing only two individual national econometric models compared with some thirty one in the full blown LINK system, we could not expect to duplicate perfectly the results of a full LINK world trade simulation. Our results, however, were remarkably close to LINK's and more importantly, were internally consistent, establishing the validity of our linkage mechanism and the usefulness of MODEL as an econometric modelling tool. Most importantly, we obtained the simulation results in which we were interested, describing trade between France and Spain.

A block diagram showing the functional organization of the linked model is shown in Figure 7-1. The model consists of four modular subspecifications combined into a single specification. Our intent has been to employ modularity to a high degree, so that logically independent models could be developed and tested separately prior to integration and then simply added to the linked model to participate in world wide trade with little or no modification to the individual specifications. Such modularity enhances maintainability, configuration control, readability and understandability of the overall model. Our initial world trade model consists of the modules FRANCE, SPAIN, ROW and LINKAGE. The simple ROW model is described first below, followed by a description of the national econometric models

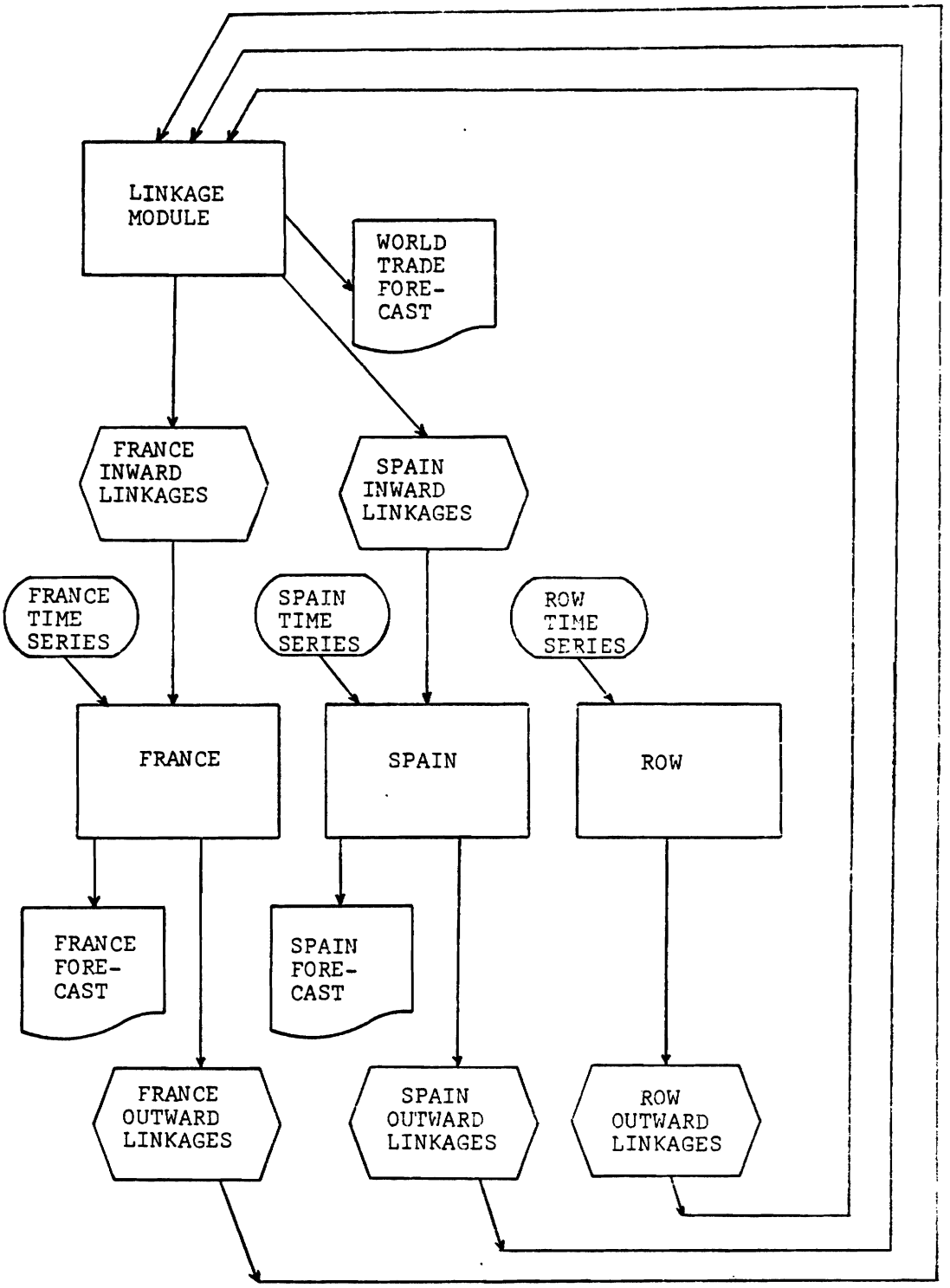


Figure 8-1. Block Diagram Showing Functional Organization of the MODEL Linked World Trade Model.

and finally a presentation of our novel linkage model.

7.1.1 ROW

Because our world trade model in this initial research stage has only two relatively small national econometric models, successful simulation of international economic behavior calls for a rest of the world model which accounts for a significant portion of international trade. In order to represent such a large amount of trade without introducing a brand new dynamic model which could adversely affect the stability of our two-nation economic community, and in order to facilitate reasonable duplication of Project LINK forecasts with this relatively small configuration, we have implemented a wholly exogenous, passive ROW model.

Data associated with the ROW model consist only of a source file, RTIM_SER, and a number of target variables giving the import prices (PM) and amount or value of exports (VX) accounted for by the rest of the world. Assertions in this model are simple equations exogenously relating the ROW target variables to their corresponding records in the RTIM_SER time series file. Data values in the RTIM_SER file were obtained by subtracting simulation solution values for FRANCE and SPAIN trade in the LINK system from the LINK values of total world trade, so that ROW values represent the balance of world trade after accounting for FRANCE and SPAIN. In this way, our MODEL sumulation can enjoy the dynamic stability afforded by exogenous values of such large

portions of trade and can hope to approximately duplicate LINK's results for comparison and validation purposes. This allows us to more thoroughly test and verify the correctness and consistency of our model.

7.1.2 FRANCE AND SPAIN

The models for FRANCE and SPAIN are closely similar to one another in terms of their organization, including source and target file organizations, assertion formats, etc. Recall that, after the model for SPAIN was successfully generated using the new MODEL facilities for sets of simultaneous equations, a model for FRANCE was patterned after the completed Spanish model. Thus, in the discussion below we will only describe the Spanish model, leaving it up to the reader to apply the principles given below to the French model presented in Appendix IV in the full model listing. In the case of many data names in the specifications, similarity is so great between the two models that the only difference is the first letter of the data or file names, S for SPAIN and F for FRANCE.

The Spanish model references three distinct source files and produces two separate target files in the course of its simulation. SCOEFF is a file containing statistically derived coefficients used in equations describing the Spanish economy. STIM_SER is a file which contains needed time series data for Spain from Project LINK's international data base. SCONS_ADJ is a file of

constant adjustment values for the Spanish model. Constant adjustment values are small numerical factors used in the equations for Spain which help to fine tune the performance of various econometric sectors. SS, for Spanish Solution, is a file into which the results of our dynamic simulation of Spain are written. It consists of a column of values for each variable, one per solution period, headed by the corresponding variable's name. SSTAT, the final file in the Spanish model, gives a summary of the performance of the Spanish economy over the period of simulation, allowing an economist to see at a glance the solution values of some key economic indicatory and their percentage changes on an annual basis.

Aside from the assertions which merely produce and format the two output reports, which are trivial and left to the reader to peruse in his leisure, the assertions in the Spanish model can be grouped into four classifications, exogenous, inward linkage, endogenous and outward linkage. These are described below in the same order in which they are presented in the actual MODEL language specification in Appendix IV.

Exogenous Assertions - We have already seen how values for some variables which are difficult to predict must be supplied exogenously. Government spending and the currency exchange rate are two such variables, for example. Time series data to define these exogenous variables, and to provide exogenous values for all the simulation variables in

the periods before the simulation becomes active, were supplied from the Project LINK data base. These exogenous assertions merely relate the exogenous variables to their corresponding historical data records in the STIM_SER time series data file.

Inward Linkage Equations - In the block diagram of Figure 7-1 and in previous discussions of linkage in econometrics, we have seen that the value of a country's exports and the price of its imports must be derived either directly or indirectly from factors outside of that nation's economy, factors determined by the nation's international trading partners. The inward linkage equations relate internal economic variables representing these trade quantities to the externally available values of them as determined by the linkage model. These variables are then used within the body of the model to perform trade and other sectoral calculations. Among the tasks performed by the inward linkage equations are conversion of linkage variables expressed in international monetary units (US DOLLARS) to units of local currency, and sometimes disaggregating trade variables into various internal commodity groupings.

Endogenous Equations - Equations supplying the defining relationships for endogenous target variables comprise the bulk of the model's mathematical substance. These equations require computer solution by iterative methods such as the Gauss-Seidel procedure. Coefficients in these equations were derived from careful statistical analyses correlating

measured economic performance with behavior predicted by the model's theoretical relationships. Constant adjustment values are added to fine tune the model equations in certain sectors, where necessary. The format of these equations is the familiar IF-THEN-ELSE format presented in the description of the four equation Spanish model in Section IV of this report. In simulation periods before the maximum number of lags in the model equations has been attained, solution values for all simulation variables must be obtained exogenously, from the time series data file. This is so that after this period, when the equations are "turned on" and become dynamically active, all references to lagged variables will be satisfiable. The values of these lagged variables will have been read in during the initial exogenous periods of the simulation.

Outward Linkage Equations - After the bulk of the model has been solved iteratively in the endogendous equations described above, the trade variables which the Spanish model can determine are fed back out into the linkage model for further calculation. The value of imports and the price of exports are such quantities, and can be used by the linkage model to help determine the inward linkage quantities of Spain's international trading partners. Similarly to inward linkage equations, the outward linkage equations must perform currency conversion, but this time in reverse, from local currency units to US DOLLARS. Outward linkages are supplied in several important commodity categories. Although these equations appear in the individual national

models, they actually define trade variables which belong to the linkage model, and which are described in the file and data description statements of that model. These equations define the linkage variables in terms of the local model's economic variables. This performs a feedback function essential to the operation of the world trade model.

7.1.3 THE LINKAGE MODEL

The linkage model serves to disaggregate international trade among the participating national and regional econometric models. It accepts as "inputs" values determined in the outward linkage sections of constituent models and produces as "outputs" disaggregated trade patterns which redistribute the trade values among the trading nations' inward linkage variables. Various linkage mechanisms have been employed in econometrics over the years. Ghana, Hickman, Lau and Jacobson (1978) have reported on alternative approaches to linkage of national econometric models. The method described in the following discussion is based on the method of Constant Value Shares (CVS) and was developed in cooperation with S. Fardoust of Project LINK and the University of Pennsylvania's Department of Economics.

The method of constant value shares assumes that nominal trade shares among participating nations remain constant throughout the period of simulation. These nominal trade shares are available in Trade Share Matrices (TSM)

which, for each nation, give the proportion of its exports in the import markets of its international trading partners. Trade Share Matrices in our linkage model are 3 x 3, with FRANCE, SPAIN and ROW represented by one row and one column each. If aij is the element in the i-th row and j-th column of such a matrix, then it corresponds to the share of country i in the import market of country j.

Imports, exports and other behavior of various commodity groups are usually modelled separately according to the structure of their corresponding economic sectors. Trade Share Matrices, for example, are distinct for each individual commodity group. For the purposes of most of our calculations, the commodity groups are assigned numbers which form a part of the variable name representing some aspect of that commodity group. These digits, in conformance with Standard International Trade Classification (SITC) goods trade, are

0,1	food, beverages and tobacco
3	mineral fuels
2,4	other basic materials
5-9	manufactures and semi-manufactures.

In addition, the digits 0-9 are used to indicate total trade, i.e. the sum of trade in all the above SITC commodity groups.

In the equations that follow, and in our MODEL language specification of this model, we adhere to the following conventions in the naming of linkage variables:

X - Exports

M - Imports

F - FRANCE (French)

S - SPAIN (Spanish)

R - ROW (Rest of World)

TW - Total World

P - Price

V - Value

k - Commodity Group.

Thus, for example, TWVXK represents the total world value of exports in commodity group k; MFk represents real imports in France in commodity group k; and VXSk represents the value of exports from Spain in commodity group k. We will also use the letters i and j to represent the nations in our econometric model, where in the above notation i and j can take the values S, F or R. Xik, for example, represents real exports in country i for commodity group k; XFik represents real exports from France to country i for commodity group k. The meaning of further combinations of these symbols in this notation should become clear as the presentation progresses.

The first equations in our linkage model use the trade share matrices to determine each model's exports as a share of its trading partners' imports for each commodity group. Recall that the value of imports is defined for SITC groups in each model's outward linkage equations. Thus,

VXijk = aijk * VMjk.

That is, the value of exports from country i to country j for commodity group k is equal to the product of the market share of country i in country j's import market and the total value of imports of country j in commodity group k.

Next, we calculate each model's total value of exports:

$$VXik = \sum_{j} VXijk.$$

That is, the total value of country i's exports in commodity group k is equal to the sum of country i's exports to each of its j trading partners for commodity group k.

The world totals are

$$TWVXk = \sum_{i} VXik.$$

That is, total world value of exports in each commodity group is equal to the sum of export values in all i participating models in that commodity group.

Next, real imports in each model are calculated by:

$$Mjk = \sum_{i} (VXijk/PXik).$$

That is, the total volume of real imports for each country j is equal to the sum of export volumes to country j by each of its i trading partners. The value of exports from country i to country j in commodity group k, VXijk, divided by the price of those exports, PXik, is equal to the real volume of those exports.

Next, the price of imports for each country j is given

by:

$$PMjk = \left(\sum_{i} VXijk \right)/Mjk.$$

That is, the import price of country j for commodity group k is equal to the total value of all exports to country j from all its i trading partners, divided by the real volume of imports of country j. The total value of all exports to j, Ei VXijk, represents the total value of j's imports; and the total value of imports divided by the volume of imports gives the price of imports for each commodity group.

Next, we can calculate the real export volumes of each country by:

$$Xjk = VXjk/PXjk,$$

where export volume is simply equal to the value of exports for each commodity group divided by the price of those exports.

Finally, the remaining total world variables are calculated by:

$$TWXk = \sum_{j} Xjk$$

and

$$TWPXk = TWVXk/TWXk.$$

Here, total world volume of exports in each commodity group k is equal to the sum of export volumes of all the countries in that commodity group; and total world price of exports

in any commodity group simply equals the total world value of exports in that commodity group divided by the total world volume of exports in that commodity group.

Taken together, these equations completely model international trade among our participating econometric models. They provide a transformation from each country's outward linkages to its inward linkages. In some of the equations, sums of values over a country's trading partners must be evaluated. For these purposes, each model's trading partners are:

country	trading partners
SPAIN	FRANCE, ROW
FRANCE	SPAIN, ROW
ROW	FRANCE, SPAIN, ROW

Note that ROW, unlike the other models, also trades with itself. This is because the ROW model represents an amalgam of individual models and their associated international trade.

In all the above equations, it is also implied that the relations hold only when all the values referenced refer to the same period of simulation. All calculations in the linkage model are performed in a standard currency unit, the US DOLLAR.

7.2 THE SIMULATION CALCULATIONS

One possible way to solve the linked world trade model

described above is to treat the entire model simply as a list of equations and to perform huge Gauss-Seidel iterations over the entire list of equations until the system converges. This method, while conceivable, is inefficient in terms of computation time and would soon grow extremely impractical as more equations were added to explicitly model more individual national and regional economies. The full Project LINK system, for example, contains some twenty thousand equations. Iterative recalculation of these equations in one enormous loop would be extremely costly in terms of both memory and CPU time.

In order to maintain a high degree of modularity, and to take advantage of the improved computational efficiency of solving simultaneous equations iteratively in tightly coupled units, we have organized the world trade model in a nested block configuration. The linkage model comprises the outer level iterative block, and the individual models comprise smaller nested blocks on an inner level. Thus, for each iteration of the linkage model, the individual models are completely and independently solved in nested Gauss-Seidel procedures, using inward linkage values fed in by the current iteration values of the outer block. As each of the nested models is solved, it communicates outward linkages to the central linkage model which, at its next iteration, recalculates the world trade distributions for feedback once again into the participating models' inward linkage equations. This process continues until the linkage model and the constituent models within it all satisfy their

respective convergence criteria simultaneously, or until some maximum number of iterations has been reached without achieving convergence. Then either the model has been solved for a given period, or appropriate error messages warning of divergence are generated, respectively.

The advantages of such a nested approach to the linked model organization and computation include:

Savings of Memory and CPU Time - More efficient computation results from smaller, localized iterations of tightly coupled sets of equations. Moreover, recall that automatic dependency analysis by the MODEL processor leads to even smaller iterations as equations which do not need to be iteratively recalculated are extracted from the simultaneous blocks and removed to higher levels of nesting. This further reduces the size and processing requirements of the computation.

Ease of Integration Testing - It is straightforward, using the nested approach, to experiment with the convergence criteria, maximum number of iterations and initial parameter values for individual country models. This leads to increased understanding of the behavior of the constituent models and allows easy integration of models in different stages of development, which might require stricter or looser convergence criteria depending on their respective stages of development.

Ease of Model Updating - Data for individual models can

be updated separately, and changes in the equations of individual models are simple and limited in the scope of their effects. Coefficients, time series and constant adjustment file changes to individual models need not affect the data files of other models, insulating them from possible errors in the new data. Changes in equations can be seen immediately in the computation results of the individual models and the number of local iterations required for each of the nested models to converge.

Other Advantages of Modularity - Increased modularity is accompanied by many other advantages, including better readability, understandability, maintainability and greater ease of configuration control. Individual models can be altered, removed, added or replaced with maximum convenience and minimum impact on the remaining participating models.

7.2.1 RESULTS OF THE SIMULATION

Using data supplied from the Project LINK international econometric database, we used our linked world trade model to simulate the period from 1978 to 1982, producing one solution per year of the simulation period. Five output reports were generated in simulation, shown in Figures 7-2 through 7-5. Figure 7-2 shows the solutions for SPAIN, Figure 7-3 for FRANCE and Figure 7-4 for the trade and linkage variables, including ROW solutions. Figure 7-5 presents two summary reports, one for each of the two dynamic constituent national models, SPAIN and FRANCE,

SPAIN: JOINT LINK/MODEL ECONOMETRIC RESEARCH GROUP

	CONS	INV	II	EX	IM	GOV	GNP	EMP	PGDP	PIM	PEX	WR	URATE
1978	26.1980	7.7440	0.9770	6.8010	6.2270	3.6270	39.1260	12614.9450	389.4330	304.3650	324.9940	587.3250	8.7890
1979	25.7350	7.6630	1.2550	6.7250	6.0530	3.5530	38.8860	12711.2000	365.3590	285.1390	310.2050	550.6200	7.9690
1980	25.5889	6.5775	0.9487	6.0445	6.8965	3.7010	35.9641	11635.2829	436.9287	350.7765	369.2306	678.5159	10.2495
1981	26.2742	7.9618	0.5892	6.3589	7.1632	3.8200	37.8409	11826.6205	481.0492	399.2781	406.8212	756.2552	11.5514
1982	26.9460	8.2543	0.6120	6.7376	7.3754	3.9800	39.1545	11839.4736	530.5697	465.6303	447.0435	841.3483	12.3019

	T	SSC	SSB	DT	IT	CONP	GDFC	NATINC	IMC	EXC	CURBAL	EXRATE	IM01C
1978	79.5000	1624.6880	1253.3410	900.0050	1026.1750	7342.5160	15743.0250	13457.6650	1894.4420	2211.9550	317.4620	69.3170	150.3020
1979	79.0000	1516.5340	1155.6060	815.6050	959.2260	6875.4770	15207.2100	12573.4500	1717.2070	2086.5370	369.2290	67.1350	138.5780
1980	80.0000	1760.3998	1473.4861	945.7947	1004.7076	7928.6502	15713.7474	14008.8314	2419.1445	2231.8145	-187.3300	71.5000	194.1459
1981	81.0000	1995.3670	1653.1652	1151.9160	1168.1163	8943.9432	18203.3346	16191.2005	2860.1091	2586.9355	-273.1736	73.0000	235.2693
1982	83.0000	2230.7704	1840.0853	1378.5247	1335.7981	9961.1209	20774.1913	18485.5824	3434.2101	3012.0007	-422.2094	75.0000	274.4145

	IM24C	IM3C	IM59C	SUB	IM09C	IM01	IM24	IM3	IM59	PIM01	PIM24	PIM3	PIM59
1978	172.6030	374.4880	1008.1650	467.1990	1700.0610	0.5960	0.6910	0.5080	3.6950	301.9790	299.2910	726.3770	322.1890
1979	158.8570	351.7850	895.4320	401.8990	1533.6540	0.5910	0.6850	0.5050	3.5320	334.4000	331.8990	675.7000	353.2000
1980	327.2530	497.2613	1212.7614	532.5000	2231.4216	0.7069	0.9927	0.4258	4.0934	274.6441	329.6596	1167.6284	296.2724
1981	398.2629	593.1590	1415.5879	653.1000	2642.2791	0.7755	1.0475	0.4018	4.2235	303.3776	360.2033	1476.2544	335.1694
1982	474.9074	804.8590	1631.6866	816.3990	3185.8675	0.8189	1.0829	0.4311	4.3014	335.1014	438.5515	1866.9891	379.3385

	PIM09	IMSERC	IMSER	PIMSER	EX01C	EX24C	EX3C	EX59C	EX09C	EXSERC	EX01	EX24	EX3
1978	309.1550	194.3860	0.7390	314.3990	276.5640	87.3430	58.7730	903.0110	1320.1480	891.8110	0.9080	0.2980	0.1450
1979	288.5750	183.5630	0.7290	351.8000	259.5530	79.5290	57.7850	879.5800	1266.3590	820.0670	0.9040	0.3340	0.1370
1980	356.8186	187.7229	0.6777	277.0000	202.8699	68.7020	54.0981	878.6582	1204.5282	1027.2863	0.6316	0.1817	0.1365
1981	409.7636	217.8300	0.7149	304.7000	228.9384	80.5000	69.7364	1026.9311	1406.1059	1180.8296	0.6480	0.1760	0.1586
1982	480.2115	248.3426	0.7411	335.1000	259.0019	94.4767	89.8602	1202.4475	1645.7863	1366.2144	0.6665	0.1901	0.1901

	EX59	EX09	EXSER	PEX01	PEX24	PEX3	PEX59	PEXSER	PEX09	IM09	CURACC	NATDIN	WTRAN
1978	3.2570	4.5610	3.2400	354.0000	349.6230	369.7000	322.1060	330.2860	324.6370	5.4870	430.5620	13470.7660	63.0990
1979	3.7290	5.0050	3.7200	386.8000	339.8700	348.8000	322.1550	301.5020	316.1130	5.3150	532.3290	12536.5500	63.1000
1980	2.3327	3.2845	2.7600	321.2000	378.1069	390.6000	376.7558	372.2052	366.7310	6.2188	-124.2300	14071.9314	63.1000
1981	2.5363	3.5189	2.8400	353.3000	457.3869	439.7000	404.6934	415.7851	399.5867	6.4483	-210.0736	16257.3005	63.1000
1982	2.7509	3.7976	2.9400	388.6000	496.9847	472.7000	437.1106	464.6988	433.3753	6.6343	-359.1094	18548.6824	63.1000

	STBL	XSL	MSL	TRANL	SCRL	GNPL	PGNPL	SEXRL
1978	-4.1230	12.8950	2.8020	2.6880	3.5320	56.1960	4.4110	0.0140
1979	-2.0950	12.2170	2.7350	2.9790	5.4080	55.8020	3.7920	0.0150
1980	-13.1462	14.2792	2.6093	2.7800	-4.2563	51.6093	4.2322	0.0139
1981	-15.4315	16.0592	2.9624	2.7200	-5.0547	54.3025	4.5590	0.0136
1982	-18.7403	18.1706	3.3029	2.6600	-6.5326	56.1875	4.9174	0.0133

Figure 7-2. Solutions of the Spanish Model.

FRANCE: JOINT LINK/MODEL ECONOMETRIC RESEARCH GROUP

	ZVM	VM	ZLCH	CH	ZWC1	WC1	ZWC2	WC2	ZPVENF	PVENF	ZPVAD	PVAD	ZPXTOT
1978	0.0080	34.5740	0.0500	1426.8340	0.1170	546.2080	0.1150	0.5040	0.0380	2.3190	0.0530	251.0980	0.0690
1979	-0.0130	34.1060	0.0500	1600.9350	0.1170	610.3750	0.1160	0.5620	0.0980	2.5450	0.0800	271.2170	0.1760
1980	0.1116	37.9022	0.0470	1763.9131	0.1495	701.6260	0.1505	0.6465	0.1261	2.8659	0.1033	299.2337	0.1769
1981	0.0845	41.1049	0.0500	2001.5834	0.1201	785.8912	0.1179	0.7227	0.0547	3.0226	0.0446	312.5795	0.0103
1982	0.0299	42.3339	0.0400	2194.6925	0.1153	876.5044	0.1148	0.8056	0.0985	3.3203	0.0780	336.9607	0.1168

	PXTOT	ZPCTOT	PCTOT	RDISV	ZCMV	CMV	VST	ZLXTOT	XTOT	CM	DFCUM	ZLEMTOT	EMTOT
1978	2.2090	0.1390	2.7680	1563.7670	0.1210	1238.4990	-2.9950	0.0870	155.4660	487.1660	82.1970	0.0040	152.6040
1979	2.3900	0.1260	3.1230	1755.6870	0.1200	1442.9340	0.3830	0.0370	160.2790	498.7920	84.5820	0.0550	160.4360
1980	2.7950	0.1452	3.5764	1993.0072	0.1469	1654.9010	-2.4578	0.1753	166.1467	500.3933	65.1075	0.0197	163.5986
1981	2.8794	0.0710	3.8303	2408.3595	0.0932	1809.1377	-1.4079	0.0456	171.9902	510.7240	86.7471	0.0171	166.3961
1982	3.1895	0.1111	4.2558	2553.6120	0.1325	2048.8484	-3.0604	0.0174	177.9129	521.0305	69.0211	0.0260	171.0551

	PIB	COTOT	ZPVM	PVM	XTOTV	EMTOTV	PVTOT	PIBV	ZPOPADC	POPADC	ZPCM	PCM	ZPOPADI
1978	726.4830	15798.4180	0.0450	2.1540	343.3960	359.1560	2.3180	1895.2630	-0.0110	21012.6410	0.0690	2.6450	0.0020
1979	743.4590	17169.1410	0.0740	1.9140	383.1380	433.5110	2.4650	2123.2480	-0.0150	20936.9960	0.0940	2.8930	-0.0020
1980	746.1651	19419.8977	0.1056	1.7161	464.3800	526.2148	2.6739	2374.6171	-0.0243	20652.6695	0.1432	3.3072	-0.0105
1981	762.5669	21342.9321	0.0271	1.7626	495.2285	555.8628	2.7686	2602.3350	-0.0152	20709.4997	0.0711	3.5423	0.0052
1982	781.2011	23036.1805	0.0754	1.8955	567.4531	656.5436	3.0469	2921.5920	-0.0140	20533.2135	0.1101	3.9323	-0.0060

	POPADI	DPLAN2	ZVENFV	VENFV	VENF	ZPOPASA	POPASA	ZREVNSV	REVNSV	REERUV	RAPDIB	PPIB	POPTOT
1978	0.0000	1.0000	0.0870	323.6780	139.5940	0.0240	16534.9530	0.1200	587.6980	1837.8690	0.6510	2.5950	53227.0000
1979	0.0000	1.0000	0.1240	363.6820	142.6940	0.0170	18646.4770	0.1150	655.3310	2069.7100	0.6460	2.8560	53471.0000
1980	0.0000	1.0000	0.0911	396.8134	138.4603	0.0129	19089.5966	0.1186	733.0532	2354.1309	0.8466	3.2092	53836.0000
1981	0.0000	1.0000	0.0255	406.9321	134.6298	0.0279	19622.1963	0.0787	790.7444	2606.9910	0.9231	3.4125	54027.1910
1982	0.0000	1.0000	0.1409	464.2688	139.8273	0.0124	19865.5115	0.1154	681.9963	2914.0843	0.8763	3.7598	54260.9680

	DURTRA	BALG63	CONTIN	BSAL68	B6263	SSALN	PBNTOT	TTDENF	BTC69	POPAAG	PRESSOC	WT	PWT
1978	89.3000	1.0000	180.0000	2.0900	2.0000	0.4300	2.3630	0.0320	2.0000	2045.0000	416.1430	493.4000	258.4610
1979	88.7000	1.0000	180.0000	2.0000	2.0000	0.4330	2.7020	0.0340	2.0000	1965.0000	474.4030	501.9610	264.5980
1980	88.3000	1.0000	180.0000	2.0000	2.0000	0.4330	3.2165	0.0360	2.0000	1888.0000	531.3500	553.5527	342.8000
1981	87.0000	1.0000	180.0000	2.0000	2.0000	0.4370	3.3406	0.0360	2.0000	1807.9000	584.5030	571.0171	389.7500
1982	87.0000	1.0000	180.0000	2.0000	2.0000	0.4390	3.8362	0.0390	2.0000	1728.8100	639.7700	569.4685	443.6600

	DEVAL4	T491	BSUAL	BTC58	CAD	CIF	EFFISC	VAD	VIF	DEVAL1	SUSTO	TTIENF	SUSTOV
1978	52.5000	30.0000	1.0000	0.0000	22.5180	4.1000	0.9000	26.1000	2.1760	52.5000	9.7900	0.1400	19.0000
1979	52.5000	31.0000	1.0000	0.0000	23.1930	4.3000	0.9000	27.2000	2.2500	52.5000	10.5000	0.1400	20.0000
1980	52.5000	32.0000	1.0000	0.0000	23.8890	4.4000	0.9000	28.2000	2.3300	52.5000	10.5000	0.1360	21.0000
1981	52.5000	33.0000	1.0000	0.0000	24.6060	4.5460	0.9000	29.2000	2.4470	52.5000	11.1230	0.1360	22.0000
1982	52.5000	34.0000	1.0000	0.0000	25.3360	4.6860	0.9000	30.1000	2.5470	52.5000	11.5630	0.1360	23.1300

	PVST	POPAAD	ZPIBPLA	PIBPLAN	NSERVL	FTBL	FCBL	PX01	PX24	PX3	PX59	M01	M24
1978	3.0400	3250.0000	0.0000	0.0570	4.2180	1.0160	5.2340	2.3270	2.1230	2.7040	2.1840	-0.6260	-0.7599
1979	3.2480	3315.0000	0.0000	0.0570	4.6460	-2.1930	2.4470	2.7060	2.5230	3.2400	2.3140	-0.6399	-0.7599
1980	3.4750	3381.0000	0.0000	0.0570	4.8090	-1.7387	3.0703	2.9305	3.0967	4.9417	2.7035	4.1039	3.1398
1981	3.7100	3446.2000	0.0000	0.0570	5.4120	-3.1642	2.2478	2.9644	3.2463	5.7520	2.7755	4.1850	3.2065
1982	3.9400	3511.6000	0.0000	0.0570	5.6899	-6.5679	-0.8760	3.2122	3.6421	7.2108	3.0641	4.3201	3.3175

	M3	M59	DFT	DFTV	DFI	PDFT	X01	X24	X3	X59	FEXRL	GDPL	PGDPL
1978	-1.6999	-0.9399	777.5900	1510.4800	643.6300	1.9400	4.6880	1.7660	0.4570	25.3060	0.2220	170.9410	2.4480
1979	-1.6999	-0.5400	777.5900	1510.4800	643.6300	1.9400	4.3690	1.6110	0.5370	26.7010	0.2320	174.9360	2.8160
1980	2.3765	21.4218	777.5900	1510.4800	643.6300	1.9400	4.0441	1.4242	0.7494	28.2142	0.2290	175.5726	3.1224
1981	2.3996	21.7973	777.5900	1510.4650	643.6290	1.9400	4.1161	1.4523	0.7543	29.3211	0.2460	179.4319	3.5666
1982	2.4314	22.4227	777.5900	1510.4650	643.6290	1.9400	4.1967	1.4845	0.7595	30.4278	0.2460	183.8166	3.9087

Figure 7-3. Solutions of the French Model.

LINKAGE VARS: JOINT LINK/MODEL ECONOMETRIC RESEARCH GROUP

	VXF01	VXF24	VXF3	VXF59	VXF09	VXS01	VXS24	VXS3	VXS59	VXS09	VXR01	VXR24	VXR3	VXR59	VXR09
1978	10.4346	4.0561	1.8643	61.8551	78.2101	2.2187	0.7861	0.3214	8.9146	12.2408	137.7844	121.0405	208.2149	783.0579	1250.0977
1979	11.8251	4.4118	2.8139	71.9230	90.9738	2.5165	0.8611	0.4915	[illegible]	14.3723	156.0951	132.1948	314.0232	917.3396	1519.6527
1980	13.4604	4.9536	4.2713	84.9250	107.6103	2.8199	0.9551	0.7523	12.2160	16.7433	176.6630	147.5596	477.7511	1075.7228	1877.6965
1981	14.8872	5.6883	5.3760	97.3344	123.2659	3.1140	1.0949	0.9485	13.9661	19.1235	195.3270	169.2866	601.4222	1231.8479	2197.8837
1982	16.4559	6.5235	6.7858	111.5119	141.2771	3.4450	1.2567	1.1957	15.9925	21.8899	216.0069	194.2649	758.4788	1411.2491	2579.9997

	MF01	MF24	MF3	MF59	MF09	MS01	MS24	MS3	MS59	MS09	MR01	MR24	MR3	MR59	MR09
1978	4.3144	2.8700	2.7604	20.5126	30.4574	0.9248	0.9317	1.1221	6.1940	9.1726	64.6734	47.4196	39.8223	351.3426	503.2579
1979	4.3902	2.8164	2.4493	21.8643	31.5102	0.7888	0.7753	0.7717	4.8138	7.1496	65.8349	45.5196	39.7577	367.3022	518.4144
1980	4.1037	3.1397	2.3765	21.4211	31.0410	0.9305	1.3058	0.5605	5.8883	8.6861	67.4995	43.7432	39.5908	382.7614	533.5949
1981	4.1849	3.2064	2.3996	21.7968	31.5877	1.0208	1.3769	0.5289	6.0598	8.9684	69.0071	44.3930	40.3891	396.4153	550.2045
1982	4.3200	3.3174	2.4314	22.4223	32.4911	1.0780	1.4255	0.5676	6.1624	9.2335	70.5791	45.0879	41.2171	410.6239	567.5080

	MMF01	MMF24	MMF3	MMF59	MMF09	MMS01	MMS24	MMS3	MMS59	MMS09	MMR01	MMR24	MMR3	MMR59	MMR09
1978	2.1416	2.4654	4.8416	2.2647	2.4997	2.1463	2.4470	4.7971	2.2621	2.5793	2.1525	2.4573	4.8127	2.2580	2.4654
1979	2.3717	2.7989	7.4571	2.5438	2.9246	2.3997	2.7987	7.3465	2.5395	3.0713	2.4022	2.7991	7.3795	2.5372	2.9144
1980	2.6302	3.1772	11.4845	2.8609	3.5226	2.6603	3.1932	11.3120	2.8698	3.4408	2.6618	3.1849	11.3445	2.8599	3.4910
1981	2.8412	3.5872	14.1975	3.1650	4.0031	2.8752	3.6033	13.9909	3.1765	3.8441	2.8765	3.5951	14.0205	3.1656	3.9608
1982	3.0682	4.0459	17.5511	3.5006	4.5502	3.1058	4.0646	17.3037	3.5158	4.4002	3.1072	4.0549	17.3220	3.5034	4.5015

	VMF01	VMF24	VMF3	VMF59	VMF09	VMS01	VMS24	VMS3	VMS59	VMS09	VMR01	VMR24	VMR3	VMR59	VMR09
1978	9.2400	7.0760	13.3650	46.4550	76.1360	1.9850	2.2800	5.3830	14.0120	23.1630	139.2130	116.5270	191.6510	793.3610	1226.8110
1979	10.3890	7.8930	18.2650	55.6200	92.1570	1.8930	2.1700	5.6710	12.2250	20.9600	158.1550	127.4150	293.3900	931.9210	1510.8780
1980	10.7940	9.9757	27.2929	61.2656	109.3482	2.4756	4.1729	6.3409	16.8991	29.6885	179.6740	139.3200	449.1370	1094.6800	1860.7200
1981	11.8904	11.5023	34.0683	68.9884	126.4494	2.9352	4.9688	7.4005	19.2493	34.5538	198.5030	159.5990	566.2730	1254.9110	2179.2850
1982	13.2549	13.4222	42.6737	78.4929	147.8437	3.3481	5.7943	9.8203	21.6660	40.6287	219.3050	182.8290	713.9580	1438.5950	2552.3890

	PXF01	PXF24	PXF3	PXF59	PXF09	PXS01	PXS24	PXS3	PXS59	PXS09	PXR01	PXR24	PXR3	PXR59	PXR09
1978	2.5620	2.3120	3.0240	2.3580	2.3950	3.0630	3.5150	3.7130	3.2950	3.2660	2.1160	2.4580	4.8420	2.2430	2.4400
1979	3.1140	2.8710	3.7860	2.6110	2.7080	2.9970	3.5280	3.6210	3.3540	3.2810	2.3520	2.7930	7.4590	2.5250	2.9310
1980	3.3284	3.4780	5.6996	3.0100	3.1252	3.1112	3.6624	3.7834	3.6493	3.5522	2.6140	3.1730	11.4900	2.8420	3.5200
1981	3.6163	3.9167	7.1267	3.3196	3.4588	3.3483	4.3347	4.1671	3.8372	3.7869	2.8240	3.5810	14.2060	3.1400	3.9990
1982	3.9192	4.3942	8.9341	3.6648	3.8316	3.6016	4.6061	4.3810	4.0512	4.0165	3.0500	4.0410	17.5640	3.4860	4.5430

	XF01	XF24	XF3	XF59	XF09	XS01	XS24	XS3	XS59	XS09	XR01	XR24	XR3	XR59	XR09
1978	4.0728	1.7543	0.6165	26.2320	32.6756	0.7243	0.2236	0.0865	2.7054	3.7398	65.1155	49.2434	43.0018	349.1118	506.4725
1979	3.7973	1.5366	0.7432	27.5461	33.6232	0.8396	0.2440	0.1357	3.1315	4.3506	66.3669	47.3307	42.0999	363.3028	519.1003
1980	4.0441	1.4242	0.7494	28.2142	34.4319	0.9063	0.2607	0.1988	3.3474	4.7132	67.5833	46.5047	41.5797	378.5090	534.1767
1981	4.1161	1.4523	0.7543	29.3211	35.6438	0.9300	0.2525	0.2276	3.6396	5.0497	69.1667	47.2735	42.3357	391.3112	550.0871
1982	4.1987	1.4845	0.7595	30.4278	36.8705	0.9565	0.2728	0.2729	3.9475	5.4497	70.8219	48.0734	43.1837	404.8333	566.9123

	TWVX01	TWVX24	TWVX3	TWVX59	TWVX09	TWX01	TWX24	TWX3	TWX59	TWX09	TWPX01	TWPX24	TWPX3	TWPX59	TWPX09
1978	150.4377	125.8827	210.4006	853.8276	1340.5486	69.9126	51.2213	43.7048	378.0492	542.8879	2.1517	2.4576	4.8141	2.2585	2.4692
1979	170.4367	137.4677	317.3286	999.7658	1624.9988	71.0038	49.1113	42.9788	393.9804	557.0743	2.4003	2.7991	7.3833	2.5376	2.9170
1980	192.9433	153.4683	482.7747	1172.8638	2002.0501	72.5337	48.1896	42.5279	410.0706	573.3218	2.6600	3.1846	11.3519	2.8601	3.4920
1981	213.3282	176.0698	607.7467	1343.1484	2340.2931	74.2128	48.9783	43.3176	424.2719	590.7806	2.8745	3.5948	14.0300	3.1657	3.9613
1982	235.9078	202.0451	766.4603	1538.7535	2743.1667	75.9771	49.8307	44.2161	439.2086	609.2325	3.1049	4.0546	17.3344	3.5034	4.5026

Figure 7-4. Solutions of the World Trade and Linkage Variables

FRANCE: DYNAMIC SIMULATION STATISTICS

			1980	1981	%CHG	1982	%CHG
AGG. DEMAND (NOM. FRANCS):	PRIV CONSUMPTION	CMV	1654.901	1809.137	9.3	2046.646	13.2
	PRIV INVESTMENT	VENFV	396.813	406.932	2.5	464.268	14.0
	GROSS DOM PROD	PIBV	2394.617	2602.335	8.6	2921.592	12.2
AGG. DEMAND (REAL FRANCS):	HSEHOLD CONS	CM	500.393	510.724	2.0	521.030	2.0
	PUBLIC CONS	CAD	23.889	24.606	3.0	25.336	2.9
	HSEHOLD INVSTMT	VM	37.902	41.104	8.4	42.333	2.9
	INDUSTRY INVSTMT	VENF	138.460	134.629	- 2.7	139.827	3.8
	PUBLIC INVSTMT	VAD	28.200	29.200	3.5	30.100	3.0
	INVENTORY CHG	VST	-2.457	-1.407	- 42.7	-3.080	118.9
	EXPORT GOODS	XTOT	166.146	171.990	3.5	177.912	3.4
	IMPORT GOODS	BMTOT	163.598	166.396	1.7	171.655	2.7
	GROSS DOM PROD	PIB	746.165	762.566	2.1	781.201	2.4
BAL. PAY. (CUR. DOLLARS):	FOB TRADE BAL	TB	-1.738	-3.164	82.0	-6.567	107.5
KEY ECONOMIC INDICATORS :	GDP DEFLATOR	FPIB	3.209	3.412	6.3	3.739	9.5
	PRIV CONS PRICE	PCM	3.307	3.542	7.1	3.932	11.0
	PRIV INV DEFL	PVENF	2.865	3.022	5.4	3.320	9.8
	EXPORTS DEFL	PXTOT	2.795	2.879	3.0	3.169	10.7
	IMPORTS DEFL	PBMTO	3.216	3.340	3.8	3.838	14.9

Figure 7-5a. Dynamic Simulation Statistics for FRANCE.

SPAIN: DYNAMIC SIMULATION STATISTICS

			1980	1981	%CHG	1982	%CHG
AGG. DEMAND (CUR. PESETA):	GROSS NATL PROD	GDPC	15713.747	18203.334	15.8	20774.191	14.1
AGG. DEMAND (1970 PESETA):	PRIV CONSUMPTION	CONS	25.588	26.274	2.6	26.946	2.5
	PUBL CONSUMPTION	GOV	3.701	3.820	3.2	3.980	4.1
	PUBL INVESTMENT	II	6.577	7.961	21.0	8.254	3.6
	PRIV INVENTORY	INV	0.948	0.589	- 37.8	0.612	3.9
	EXPORT GOODS	EX09	3.284	3.518	7.1	3.797	7.9
	IMPORT GOODS	IM09	6.218	6.448	3.6	6.634	2.8
	GROSS DOM PROD	GDP	35.964	37.840	5.2	39.154	3.4
BAL. PAY.(CUR US DOLLARS):	FOB TRADE BAL	TB	-13.146	-15.431	17.3	-18.740	21.4
KEY ECONOMIC INDICATORS :	GDP DEFLATOR	PGDP	436.928	481.049	10.0	530.569	10.2
	EXPORTS DEFLATOR	PEX	369.230	406.821	10.1	447.043	9.8
	IMPORTS DEFLATOR	PIM	350.778	399.278	13.8	465.630	16.6
	UNEMPL. RATE (%)	URATE	10.249	11.551	12.7	12.301	6.4

Figure 7-5b. Dynamic Simulation Statistics for SPAIN.

giving values and annual percentage changes for certain key variables over the dynamic period of the simulation. The solutions in 1978-1979 are given by exogenous values, and the dynamic simulation is solved in 1980-1982. This gives all referenced lagged variables a chance to assume proper historical values, as discussed earlier in this report. Thus three years of dynamic simulation solutions were successfully completed as shown in the summary reports of Figure 7-5.

The results of our simulation were found to be internally consistent and satisfactorily close to LINK's solutions after careful review by Project LINK economists.

Computations were performed on the VAX 11/780 of the University of Pennsylvania's Department of Computer and Information Science. The system was run interactively and was quite efficient, allowing us to experiment profusely with convergence criteria, data base parameters, etc. A solution of any given year of the dynamic simulation perion required approximately five to seven iterations of the linkage model to converge, and completed in an average of six CPU seconds or less, depending on initial conditions, convergence criteria, etc. For each iteration of the central linkage section at the outer level of nesting, the inner nested models were totally solved, requiring from about ten to fifteen iterations each on the first iteration of the linkage model, to about two to four iterations after the penultimate iteration of the linkage model and just

before the system as a whole converged. Of cource, the actual number of iterations varied as the data bases and convergence criteria, etc. were adjusted.

For a full listing of the MODEL language linked world trade model, its corresponding object PL/I program and associated documentation produced by the MODEL automatic processor, see Appendix III. The next and final chapter of this report presents some conclusions and recommendations at the termination of this phase of MODEL research.

CHAPTER 8

CONCLUSION

In this report we have seen how the progress of research in econometrics has been impeded by the slow advances in econometric modelling software systems, and how this situation has given rise to requirements which can be satisfied through the use of nonprocedural computational specification tools. Modifications to the MODEL automatic software generation system fulfilling these requirements have been presented. Usefulness of these new tools has been verified through the development of a nonprocedural version of a small linked world trade model in cooperation with Project LINK, and patterned after LINK's World Trade Model.

8.1 ACCOMPLISHMENTS OF THE RESEARCH

This research has demonstrated the usefulness of nonprocedural tools in econometric modelling in a variety of ways discussed throughout this report. Perhaps the most important of these advantages are the increases in modularity, efficiency and understandability of econometric

models generated with the MODEL system.

Modularity exists in the ability to completely specify and test individual econometric models separately before concatenating them together simply and easily to form a linked trade model. Within individual models, this also refers to the ability to specify local data files modularly, without necessary regard to the organization of data in other models with which any given one is to be linked. Modularity leads to increased readability and hence understandability of the models and enhances maintainability and ease of configuration control of the system.

Efficiency of econometric models is improved via the MODEL system by its unique ability to pare down sets of equations to pure simultaneous sets, avoiding unnecessary computations, and by its intelligent scheduler, which can organize data flow and computational order so as to minimize required memory and computing time.

Understandability of econometric models is inhanced by the automatic generation of thorough documentation, the ease of experimenting with models produced and the more natural tnedency in MODEL to approach a problem from the point of view of the data and transformations involved in a computational task.

Aside from these advantages, the use of nonprocedural specification systems such as MODEL confer additional benefits upon the user. The greatest of these is the

overall lowering of required expertise in programming and computer science required of the MODEL user. He may employ natural variable names and the general language of mathematics to specify complex computational tasks. He need not know or understand the intricacies of procedural programming languages and their subtle differences in various implementations. MODEL specifications are more concise, natural and easier to write and comprehend than procedural programs which perform the same computations. Moreover, the addition to the MODEL system of specialized knowledge in numerical and statistical analysis relieves the user of the burden of specifying the use of these tools in great detail. He can simply direct the system to employ such techniques and let the automatic processor do the rest.

The success of the current modifications was confirmed by our successful generation of a linked world trade model consisting of four modules: a model of France, one of Spain, a model for the rest of the world and a central linkage model simulating trade among the other three. The resultant simulation was solved for the five years from 1978-1982, producing results both internally consistent and satisfactorily close to results of the full Project LINK World Trade Model after which it was patterned.

8.2 RECOMMENDATIONS FOR FUTURE WORK

This phase of MODEL research has been concluded. However, there are a number of intriguing possibilities for

future work in this area which been suggested by this research. Among them are:

-Expansion of the MODEL linked trade model to more dynamically model the rest of the world, including the addition of a greater number of individual country/region models. This would allow us to study even more indepth the needs of econometrics, to enable us to suggest additional MODEL improvements.

-Addition of prototype or macro-type report description facilities to the MODEL system. Some means of specifying a skeletal outline of a report format might be devised to enable shortening of individual models' output report descriptions when a standard format output report for the participating models is desired. Such a capability could be extended to handle input files as well, to be used when common data base organizations are shared by several models. This could help to further reduce the work required of a modeler in writing a specification.

-Inclusion of new solution methods for complex computations. The ability to automatically incorporate a wide variety of numerical and statistical techniques would further expand the usefulness of MODEL as a tool in the social and natural sciences and in engineering. Some possibilities are alternative solution methods for sets of simultaneous equations, single or multiple regression analysis, and optimization of objective

functions subject to constraints.

-Additional corroborative research with a variety of groups. Just as our work with the economists of Project LINK has provided insight into the broad potential applications of MODEL in economics, so too could research with professionals in other areas be fruitful.

-Modifications to the MODEL system to permit distributed computations to be performed. As communication technology and the field of distributed processing continue to progress, nonprocedural tools such as MODEL might find widespread novel application. In a field such as econometrics, the usefulness of MODEL would be expanded if economists in participating countries could create and solve simulations of world trade without leaving their own countries or shipping their models to a remote central location for linkage. This topic is the subject of a current report by Pnueli and Prywes (1981).

In conclusion, it appears that MODEL has once again proven to be a powerful and general tool for specifying computational tasks.

APPENDIX I

REFERENCES

IN

ECONOMICS AND COMPUTER SCIENCE

REFERENCES

(COMPUTER SCIENCE)

Ashcroft, E. A. and Wadge, W. W. (1977), "LUCID, A Nonprocedural Language With Iteration," Communications of the ACM, Vol. 20, No. 7, pp. 519-526.

Gana, J.L. (1978), "Use and Extension of an Automatic Program Generator for Model Building in Social and Engineering Sciences," Ph.D. Dissertation in Computer and Information Science, University of Pennsylvania, Philadelphia, Pa., 19104.

Leavenworth, B. M. and Sammet, J.E. (1974), "Overview of Nonprocedural Languages," Proceedings of the Symposium on Very High Level Languages, SIGPLAN notices, ACM.

MODEL II - Automatic Program Generator, User Manual (1978), Revision of Version 3, Contract TIR-77-41, Office of Planning and Research, Internal Revenue Service, Washington, D.C.

Pnueli, A., Lu, K. and Prywes, N. (1980), MODEL Program Generator: System and Programming Documentation, Technical Report, Moore School of Electrical Engineering, University of Pennsylvania.

Pnueli, A. and Prywes, N. (1981), Distributed Processing in the MODEL System- With an Application to Econometric Modelling, Design Report, Moore School of Electrical Engineering, University of Pennsylvania.

Pnueli, A. and Prywes, N. (1980), Operations on Arrays and Data Structures , Design Report, Moore School of Electrical Engineering, University of Pennsylvania.

Prywes, N., Pnueli, A. and Shastry, S. (1979), "Use of a Nonprocedural Specification Language and Associated Program Generator in Software Development," ACM Transactions on Programming Languages and Systems, Vol. 1, No. 2, PP. 196-217.

Prywes, N. and Pnueli, A. (1981), "Compilation of Nonprocedural Specifications into Computer Programs," Technical Report, Moore School of Electrical Engineering, University of Pennsylvania.

Shastry, S. (1978), Verification and Correction of Nonprocedural Specification in Automatic Generation of Programs, Ph.D. Dissertation in Computer and Information Science, University of Pennsylvania, Philadelphia, Pa., 19104.

REFERENCES
(ECONOMICS)

Ball, R.J., ed. (1973), The International Linkage of National Models. Amsterdam: North-Holland.

Fardoust, S. (1981), "A Linkage Method for the LINK-MOORE Project," Working Paper, Economics Department, University of Pennsylvania.

Gana, J. L., Hickman, B.G., Lau, L. J. and Jacobson, L.R. (1978), "Alternative Approaches to Linkage of National Econometric Models," in Sawyer, J.A., ed. Modelling the International Transmission Mechanism, PP. 9-44. Amsterdam: North Holland.

Johnson, K. N. and Klein, L. R. (1974), "Stability in the International Economy: The LINK Experience," in Ando, A., Herring, R. and Marston, R., eds., International Aspects of Stabilization Policies, pp. 147-188. Boston: Federal Reserve Bank of Boston.

Klein, L. R. (1975), "The LINK Model of World Trade with Application to 1972-1973," in Kenen, P., ed., International Trade and Finance. Cambridge: Cambridge University Press.

Klein, L. R. (1976), "Five Years Experience of Linking National Econometric Models and of Forecasting International Trade," in Glejser, H., ed., Quantitative Studies of International Economic Relations. Amsterdam: North-Holland.

Klein, L. R. (1977), "Project LINK." Center of Planning and Economic Research, Lecture Series, 30, Athens 1977.

Moriguchi, C. (1973), "Forecasting and Simulation of the World Economy," American Economic Review, Papers and Proceedings (May), pp. 402-409.

Waelbroeck, J. L., ed. (1975), The Models of Project LINK. Amsterdam: North-Holland.

APPENDIX II

MODEL OUTPUT REPORTS

FROM COMPILATION OF THE

SHORT SPANISH MODEL

```
/*****************************************************************/
/*                  SPAIN MODULE SPECIFICATION                   */
/*****************************************************************/

  1    MODULE: SPAIN;            /*         SPANISH TRADE MODEL          */
  2
  3    SOURCE FILE: SIMDEF;     /* SIMULATION PARAMETER DEFINITION FILE  */

/*****************************************************************/
/*                       FILE DESCRIPTIONS:                      */
/*****************************************************************/


/*****************************************************************/
/*                DESCRIPTION OF SIMDEF     FILE                 */
/*****************************************************************/

  4    SIMDEF IS FILE (INPUTREC);
  5      INPUTREC IS RECORD(BEG_YR,PD_SIM,LAG);
  6        BEG_YR   IS FIELD (PIC'9999');  /* YEAR OF START OF SIMULATION          */
  7        PD_SIM   IS FIELD (PIC'999');   /* NUMBER OF PERIODS IN OUR SIMULATION  */
  8        LAG      IS FIELD (PIC'999');   /* MAXIMUM NUMBER OF LAG PERIODS IN MODEL*/
  9
 10    SOURCE FILE:  COEFF;      /* TRADE MODEL EQUATION COEFFICIENTS FILE  */

/*****************************************************************/
/*                DESCRIPTION OF COEFF      FILE                 */
/*****************************************************************/

 11    COEFF IS FILE (CO_REC(73));
 12      CO_REC IS RECORD(A);
 13       A IS FIELD (DEC FLOAT(10));
 14
 15    SOURCE FILE:  TIM_SER;    /* TIME-SERIES (HISTORICAL) DATA FILE  */

/*****************************************************************/
/*                DESCRIPTION OF TIM_SER    FILE                 */
/*****************************************************************/

 16    TIM_SER IS FILE (TS_REC(11));
 17      TS_REC IS RECORD(VNAME,VNUM,NUM_PDS,TS_DATA(1:20));
 18        VNAME    IS FIELD (CHAR(4));     /* NAME OF MODEL VARIABLE         */
 19        VNUM     IS FIELD (PIC'9999');   /* NUMERIC IDENTIFIER OF VARIABLE */
 20        NUM_PDS  IS FIELD (PIC'9999');   /* NUMBER OF PERIODS OF TIME      */
 21                                         /* SERIES DATA FOR THIS VARIABLE  */
 22       TS_DATA IS FIELD (PIC'S99.V999'); /* TIME SERIES DATA VALUE         */
 23      SIZE.TS_DATA = NUM_PDS;    /* ONE DATA VALUE PER PERIOD PER VARIABLE   */
 24
```

```
 25     TARGET FILE: SOLUTION;   /* SIMULATION SOLUTION FILE */

/*********************************************************************/

/*                 DESCRIPTION OF SOLUTION   FILE                    */

/*********************************************************************/

 26     SOLUTION IS FILE (SOL_GRP(*));
 27       SOL_GRP IS GROUP(HDR_REC,SOL_REC);   /* EACH SOLUTION GROUP CONTAINS */
 28                                            /*  A HEADER AND A BODY         */
 29       HDR_REC IS RECORD(SM_PD_ID,SM_YR_ID);
 30         SM_PD_ID    IS FIELD(PIC'9999');      /* SOLUTION PERIOD NUMBER  */
 31         SM_YR_ID  IS FIELD(PIC'BB9999');      /* SOLUTION YEAR           */
 32
 33       SOL_REC IS RECORD(II,EX,GOV,IMSER,IM01,IM24,IM3,CONS,INV,IM,GDP);
 34        (CONS,INV,II,EX,IM,GOV,GDP,IMSER,IM01,IM24,IM3) ARE FIELD (PIC'BB99.V(6)9');
 35
 36       SIZE.SOL_GRP = PD_SIM;      /* ONE SOLUTION RECORD FOR EACH PD. OF SIM. */
 37       T IS SUBSCRIPT;             /* T (TIME) IS A COUNTER OF SIMULATION PDS. */
 38
 39     /* NOW WE DEFINE VALUES FOR EXOGENOUS VARIABLES IN OUR MODEL  */
 40
 41     II   (T) = TS_DATA( 3,T);  EX   (T) = TS_DATA( 4,T);  GOV  (T) = TS_DATA( 6,T);
 42     IMSER(T) = TS_DATA( 8,T);  IM01 (T) = TS_DATA( 9,T);  IM24 (T) = TS_DATA(10,T);
 43     IM3  (T) = TS_DATA(11,T);
 44
 45     /* HERE, WE DEFINE VALUES AMONG OUR ENDOGENOUS VARIABLES  */
 46
 47     CONS(T)  = IF (T>LAG) THEN A(1) + A(2)*GDP(T) + A(3)*CONS(T-1)
 48                ELSE           TS_DATA(1,T);
 49     INV(T)   = IF (T>LAG) THEN A(4) + A(5)*GDP(T) + A(6)*GDP(T-1) + II(T)
 50                ELSE           TS_DATA(2,T);
 51     IM(T)    = IF (T>LAG) THEN IM01(T)+IM24(T)+IM3(T)+A(61)+A(62)*GDP(T)+IMSER(T)
 52                ELSE           TS_DATA(5,T);
 53     GDP(T)   = IF (T>LAG) THEN CONS(T) + INV(T) + EX(T) + GOV(T) - IM(T)
 54                ELSE           TS_DATA(7,T);
 55
 56     /* FINALLY, WE PRESENT EQUATIONS FOR A SIMPLE SOLUTION GROUP HEADER */
 57
 58     SM_PD_ID(T) = T;                    /* NUMBER OF SOLUTION PERIOD  */
 59     SM_YR_ID(T) = BEG_YR + T - 1;       /* YEAR OF SIMULATION RESULTS */
```

CROSS REFERENCE AND ATTRIBUTES REPORT

NAME	STATEMENT DESCRIPTION	NESTING LEVEL	ATTRIBUTES	REFERENCES
A	13		FIELD, DECIMAL FLOATING(10) IN FILE COEFF	51, 49, 47, 12
AASS230	23		ASSERTION	
AASS360	36		ASSERTION	
AASS410	41		ASSERTION	
AASS411	41		ASSERTION	
AASS412	41		ASSERTION	
AASS420	42		ASSERTION	
AASS421	42		ASSERTION	
AASS422	42		ASSERTION	
AASS430	43		ASSERTION	
AASS470	47		ASSERTION	
AASS490	49		ASSERTION	
AASS510	51		ASSERTION	
AASS530	53		ASSERTION	
AASS580	58		ASSERTION	
AASS590	59		ASSERTION	
BEG_YR	6		FIELD, PICTURE'9999' IN FILE SIMDEF	59, 5
COEFF	11		FILE,SOURCE,UNSORTED	10
CONS	34		FIELD, PICTURE'BB99.V(6)9' IN FILE SOLUTION	53, 47, 33
CO_REC	12		RECORD,(1 SUB-MEMBERS), IN FILE COEFF	11
EX	34		FIELD, PICTURE'BB99.V(6)9' IN FILE SOLUTION	53, 41, 33
GDP	34		FIELD, PICTURE'BB99.V(6)9' IN FILE SOLUTION	53, 51, 49, 47, 33
GOV	34		FIELD, PICTURE'BB99.V(6)9' IN FILE SOLUTION	53, 41, 33
HDR_REC	29		RECORD,(2 SUB-MEMBERS), IN FILE SOLUTION	27
II	34		FIELD, PICTURE'BB99.V(6)9' IN FILE SOLUTION	49, 41, 33
IM	34		FIELD, PICTURE'BB99.V(6)9' IN FILE SOLUTION	53, 51, 33
IM01	34		FIELD, PICTURE'BB99.V(6)9' IN FILE SOLUTION	51, 42, 33
IM24	34		FIELD, PICTURE'BB99.V(6)9' IN FILE SOLUTION	51, 42, 33
IM3	34		FIELD, PICTURE'BB99.V(6)9' IN FILE SOLUTION	51, 43, 33
IMSER	34		FIELD, PICTURE'BB99.V(6)9' IN FILE SOLUTION	51, 42, 33

```
INPUTREC          5    RECORD,( 3 SUB-MEMBERS),          4
                       IN FILE SIMDEF
INV              34    FIELD,                           53,  49,  33
                       PICTURE'BB99.V(6)9' IN
                       FILE SOLUTION
LAG               8    FIELD, PICTURE'999' IN           53,  51,  49,  47,  5
                       FILE SIMDEF
NSTGNM1           4    DISK NAME                         4
NSTGNM2          11    DISK NAME                        11
NSTGNM3          16    DISK NAME                        16
NSTGNM4          26    DISK NAME                        26
NUM_PDS          20    FIELD, PICTURE'9999' IN          23,  17
                       FILE TIM_SER
PD_SIM            7    FIELD, PICTURE'999' IN           36,   5
                       FILE SIMDEF
SIMDEF            4    FILE,SOURCE,UNSORTED              3
SIZE                   RESERVED WORD                    36,  23
SM_PD_ID         30    FIELD, PICTURE'9999' IN          58,  29
                       FILE SOLUTION
SM_YR_ID         31    FIELD, PICTURE'BB9999'           59,  29
                       IN FILE SOLUTION
SOLUTION         26    FILE,TARGET,UNSORTED             25
SOL_GRP          27    GROUP,( 2 SUB-MEMBERS),          36,  26
                       IN FILE SOLUTION
SOL_REC          33    RECORD,(11 SUB-MEMBERS),         27
                       IN FILE SOLUTION
SPAIN             1    MODULE NAME
SUBSCRIPT              UNKNOWN                          37
T                37                                     59,  58,  53,  51,  49,  47,  43,  42,  42,
TIM_SER          16    FILE,SOURCE,UNSORTED             15
TS_DATA          22    FIELD, PICTURE'S99.V999'         53,  51,  49,  47,  43,  42,  42,  42,  41,
                       IN FILE TIM_SER
TS_REC           17    RECORD,( 4 SUB-MEMBERS),         16
                       IN FILE TIM_SER
VNAME            18    FIELD, CHARACTER(    4)          17
                       IN FILE TIM_SER
VNUM             19    FIELD, PICTURE'9999' IN          17
                       FILE TIM_SER
 61      $YSGEN1 IS GROUP(SIZE.TIM_SER.TS_DATA(*));
```

RANGE TABLE

RANGE NO.	RANGE DEFINITION	WHERE DEFINED
1	CONSTANT LIMIT:73	COEFF.CO_REC
2	SIZE	SOLUTION.SOL_GRP
3	CONSTANT LIMIT:11	TIM_SER.TS_REC
4	SIZE	TIM_SER.TS_DATA

	RANGE NO.
	1 2 3 4
*ASSERTION(S):	
AASS230	1
AASS410	1
AASS411	1
AASS412	1
AASS420	1
AASS421	1
AASS422	1
AASS430	1
AASS470	1
AASS490	1
AASS510	1
AASS530	1
AASS580	1
AASS590	1
*Q_NAME:(COEFF.)	
A	1
CO_REC	1
*Q_NAME:(SIZE.TIM_SER.)	
TS_DATA	1
*Q_NAME:(SOLUTION.)	
CONS	1
EX	1
GDP	1
GOV	1
HDR_REC	1
II	1
IM	1
IM01	1
IM24	1
IM3	1
IMSER	1
INV	1
SM_PD_ID	1
SM_YR_ID	1
SOL_GRP	1
SOL_REC	1
*Q_NAME:(TIM_SER.)	
NUM_PDS	1
TS_DATA	1 2
TS_REC	1
VNAME	1
VNUM	1

```
/*****************************************************************/
/*                FORMATTED REPORT                               */
/*****************************************************************/


/*****************************************************************/
/*            SPAIN      MODULE SPECIFICATION                    */
/*****************************************************************/

 1    MODULE: SPAIN;
 2    SOURCE: TIM_SER,COEFF,SIMDEF;
 3    TARGET: SOLUTION;

/*****************************************************************/
/*                    DATA DESCRIPTION:                          */
/*****************************************************************/


/*****************************************************************/
/*            DESCRIPTION OF TIM_SER                             */
/*****************************************************************/

 4    TIM_SER IS FILE(TS_REC(11))
 4    STORAGE NAME IS NSTGNM3
 4    ORG SAM;
 5      TS_REC IS RECORD (VNAME,VNUM,NUM_PDS,TS_DATA(1:20));
 6        VNAME IS FIELD (CHAR(4));
 7        VNUM IS FIELD (PIC '9999');
 8        NUM_PDS IS FIELD (PIC '9999');
 9        TS_DATA IS FIELD (PIC 'S99.V999');

/*****************************************************************/
/*            DESCRIPTION OF COEFF                               */
/*****************************************************************/

10    COEFF IS FILE(CO_REC(73))
10    STORAGE NAME IS NSTGNM2
10    ORG SAM;
11      CO_REC IS RECORD (A);
12        A IS FIELD (FLOAT DECIMAL(10));
```

```
/*****************************************************************/

/*            DESCRIPTION OF SIMDEF                              */

/*****************************************************************/

13    SIMDEF IS FILE(INPUTREC)
13    STORAGE NAME IS NSTGNM1
13    ORG SAM;
14      INPUTREC IS RECORD (BEG_YR,PD_SIM,LAG);
15        BEG_YR IS FIELD (PIC '9999');
16        PD_SIM IS FIELD (PIC '999');
17        LAG IS FIELD (PIC '999');

/*****************************************************************/

/*            DESCRIPTION OF SOLUTION                            */

/*****************************************************************/

18    SOLUTION IS FILE(SOL_GRP(*))
18    STORAGE NAME IS NSTGNM4
18    ORG SAM;
19      SOL_GRP IS GROUP (HDR_REC,SOL_REC);
20        HDR_REC IS RECORD (SM_PD_ID,SM_YR_ID);
21          SM_PD_ID IS FIELD (PIC '9999');
22          SM_YR_ID IS FIELD (PIC 'BB9999');
23        SOL_REC IS RECORD (II,EX,GOV,IMSER,IM01,IM24,IM3,CONS,INV,IM,GDP);
24          II IS FIELD (PIC 'BB99.V(6)9');
25          EX IS FIELD (PIC 'BB99.V(6)9');
26          GOV IS FIELD (PIC 'BB99.V(6)9');
27          IMSER IS FIELD (PIC 'BB99.V(6)9');
28          IM01 IS FIELD (PIC 'BB99.V(6)9');
29          IM24 IS FIELD (PIC 'BB99.V(6)9');
30          IM3 IS FIELD (PIC 'BB99.V(6)9');
31          CONS IS FIELD (PIC 'BB99.V(6)9');
32          INV IS FIELD (PIC 'BB99.V(6)9');
33          IM IS FIELD (PIC 'BB99.V(6)9');
34          GDP IS FIELD (PIC 'BB99.V(6)9');

/*****************************************************************/

/*                  INTERIM SPECIFICATION                        */

/*****************************************************************/


/*****************************************************************/

/*                  SUBSCRIPT(S) SPECIFICATION                   */

/*****************************************************************/

35    T SUBSCRIPT;
```

```
/*****************************************************************/

/*              ASSERTION SPECIFICATION                          */

/*****************************************************************/

/* ASSERTION(S) IN BLOCK(S) */
 36    BLOCK $MAINBLOCK: MAX ITER IS 100, RELATIVE ERROR IS  1.000000E-04,
 36        SOLUTION METHOD IS GAUSS_SEIDEL;
 37      SOLUTION.CONS ($LSUB1 )=
 37        IF $LSUB1 >SIMDEF.LAG
 37          THEN COEFF.A (1 )+COEFF.A (2 )*SOLUTION.GDP ($LSUB1 )
 37              +COEFF.A (3 )*SOLUTION.CONS (($LSUB1 -1 ));
 37          ELSE TIM_SER.TS_DATA (1 ,$LSUB1 );
 38      SOLUTION.INV ($LSUB1 )=
 38        IF $LSUB1 >SIMDEF.LAG
 38          THEN COEFF.A (4 )+COEFF.A (5 )*SOLUTION.GDP ($LSUB1 )
 38              +COEFF.A (6 )*SOLUTION.GDP (($LSUB1 -1 ))
 38              +SOLUTION.II ($LSUB1 );
 38          ELSE TIM_SER.TS_DATA (2 ,$LSUB1 );
 39      SOLUTION.IM ($LSUB1 )=
 39        IF $LSUB1 >SIMDEF.LAG
 39          THEN SOLUTION.IM01 ($LSUB1 )+SOLUTION.IM24 ($LSUB1 )
 39              +SOLUTION.IM3 ($LSUB1 )+COEFF.A (61 )+COEFF.A (62 )
 39              *SOLUTION.GDP ($LSUB1 )+SOLUTION.IMSER ($LSUB1 );
 39          ELSE TIM_SER.TS_DATA (5 ,$LSUB1 );
 40      SOLUTION.GDP ($LSUB1 )=
 40        IF $LSUB1 >SIMDEF.LAG
 40          THEN SOLUTION.CONS ($LSUB1 )+SOLUTION.INV ($LSUB1 )
 40              +SOLUTION.EX ($LSUB1 )+SOLUTION.GOV ($LSUB1 )
 40              -SOLUTION.IM ($LSUB1 );
 40          ELSE TIM_SER.TS_DATA (7 ,$LSUB1 );
 41    END $MAINBLOCK;
/* ASSERTION(S) FOR FILE(SOLUTION) */
 42    SOLUTION.SM_PD_ID ($LSUB1 )=$LSUB1 ;
 43    SOLUTION.SM_YR_ID ($LSUB1 )=SIMDEF.BEG_YR +$LSUB1 -1 ;
 44    SOLUTION.II ($LSUB1 )=TIM_SER.TS_DATA (3 ,$LSUB1 );
 45    SOLUTION.EX ($LSUB1 )=TIM_SER.TS_DATA (4 ,$LSUB1 );
 46    SOLUTION.GOV ($LSUB1 )=TIM_SER.TS_DATA (6 ,$LSUB1 );
 47    SOLUTION.IMSER ($LSUB1 )=TIM_SER.TS_DATA (8 ,$LSUB1 );
 48    SOLUTION.IM01 ($LSUB1 )=TIM_SER.TS_DATA (9 ,$LSUB1 );
 49    SOLUTION.IM24 ($LSUB1 )=TIM_SER.TS_DATA (10 ,$LSUB1 );
 50    SOLUTION.IM3 ($LSUB1 )=TIM_SER.TS_DATA (11 ,$LSUB1 );
 51    SIZE.SOLUTION.SOL_GRP =SIMDEF.PD_SIM ;
/* END OF ASSERTION */

/*****************************************************************/

/*              END OF FORMATTED REPORT                          */

/*****************************************************************/
```

FLOWCHART REPORT

NODE$	NAME	DESCRIPTION	EVENT
71	SPAIN	MODULE NAME	PROCEDURE HEADING
73	TIM_SER	FILE	OPEN FILE
49	SIMDEF	FILE	OPEN FILE
50	SIMDEF.INPUTREC	RECORD IN FILE SIMDEF	READ RECORD
51	SIMDEF.BEG_YR	FIELD IN RECORD SIMDEF.INPUTREC	
52	SIMDEF.PD_SIM	FIELD IN RECORD SIMDEF.INPUTREC	
53	SIMDEF.LAG	FIELD IN RECORD SIMDEF.INPUTREC	
46	COEFF	FILE	OPEN FILE
		ITERATION	FOR FOR_EACH.COEFF.CO_REC UNTIL CONSTANT LIMITS
47	COEFF.CO_REC	RECORD IN FILE COEFF	READ RECORD
48	COEFF.A	FIELD IN RECORD COEFF.CO_REC	
		END ITERATION	FOR FOR_EACH.COEFF.CO_REC
32	AASS360	ASSERTION	
79	SIZE.SOLUTION.SOL_GRP	SPECIAL NAME	TARGET OF ASSERTION: AASS360
		ITERATION	FOR FOR_EACH.TIM_SER.TS_REC UNTIL CONSTANT LIMITS
74	TIM_SER.TS_REC	RECORD IN FILE TIM_SER	READ RECORD
75	TIM_SER.VNAME	FIELD IN RECORD TIM_SER.TS_REC	
76	TIM_SER.VNUM	FIELD IN RECORD TIM_SER.TS_REC	
77	TIM_SER.NUM_PDS	FIELD IN RECORD TIM_SER.TS_REC	
31	AASS230	ASSERTION	
80	SIZE.TIM_SER.TS_DATA	SPECIAL NAME	TARGET OF ASSERTION: AASS230
		ITERATION	FOR FOR_EACH.TIM_SER.TS_DATA UNTIL SIZE.X SPECIFIED
78	TIM_SER.TS_DATA	FIELD IN RECORD TIM_SER.TS_REC	
		END ITERATION	FOR FOR_EACH.TIM_SER.TS_DATA
		END ITERATION	FOR FOR_EACH.TIM_SER.TS_REC
		ITERATION	FOR FOR_EACH.SOLUTION.SOL_GRP UNTIL SIZE.X SPECIFIED
44	AASS580	ASSERTION	
57	SOLUTION.SM_PD_ID	FIELD IN RECORD SOLUTION.HDR_REC	TARGET OF ASSERTION: AASS580
45	AASS590	ASSERTION	
58	SOLUTION.SM_YR_ID	FIELD IN RECORD SOLUTION.HDR_REC	TARGET OF ASSERTION: AASS590
56	SOLUTION.HDR_REC	RECORD IN FILE SOLUTION	WRITE RECORD
34	AASS411	ASSERTION	
61	SOLUTION.EX	FIELD IN RECORD SOLUTION.SOL_REC	TARGET OF ASSERTION: AASS411
35	AASS412	ASSERTION	
62	SOLUTION.GOV	FIELD IN RECORD SOLUTION.SOL_REC	TARGET OF ASSERTION: AASS412
36	AASS420	ASSERTION	
63	SOLUTION.IMSER	FIELD IN RECORD SOLUTION.SOL_REC	TARGET OF ASSERTION: AASS420
37	AASS421	ASSERTION	
64	SOLUTION.IM01	FIELD IN RECORD SOLUTION.SOL_REC	TARGET OF ASSERTION: AASS421
38	AASS422	ASSERTION	
65	SOLUTION.IM24	FIELD IN RECORD SOLUTION.SOL_REC	TARGET OF ASSERTION: AASS422
39	AASS430	ASSERTION	
66	SOLUTION.IM3	FIELD IN RECORD SOLUTION.SOL_REC	TARGET OF ASSERTION: AASS430
33	AASS410	ASSERTION	
60	SOLUTION.II	FIELD IN RECORD SOLUTION.SOL_REC	TARGET OF ASSERTION: AASS410
		ITERATION(GAUSS_SEIDEL)	SIMULTANEOUS BLOCK: $MAINBLOCK
40	AASS470	ASSERTION	
67	SOLUTION.CONS	FIELD IN RECORD SOLUTION.SOL_REC	TARGET OF ASSERTION: AASS470
41	AASS490	ASSERTION	
68	SOLUTION.INV	FIELD IN RECORD SOLUTION.SOL_REC	TARGET OF ASSERTION: AASS490
42	AASS510	ASSERTION	
69	SOLUTION.IM	FIELD IN RECORD SOLUTION.SOL_REC	TARGET OF ASSERTION: AASS510
43	AASS530	ASSERTION	
70	SOLUTION.GDP	FIELD IN RECORD SOLUTION.SOL_REC	TARGET OF ASSERTION: AASS530
		END ITERATION	SIMULTANEOUS BLOCK: $MAINBLOCK
59	SOLUTION.SOL_REC	RECORD IN FILE SOLUTION	WRITE RECORD
55	SOLUTION.SOL_GRP	GROUP IN FILE SOLUTION	
		END ITERATION	FOR FOR_EACH.SOLUTION.SOL_GRP
54	SOLUTION	FILE	CLOSE FILE
81	$YSGEN1	GROUP	
		END	

```
SPAIN: PROCEDURE OPTIONS(MAIN);
DCL TIM_SERS RECORD SEQL INPUT;
DCL $FSTTIM_SERS BIT(1) INIT('1'B);
DCL ENDFILE$TIM_SER BIT(1) INIT('0'B);
DCL TIM_SER_S CHAR(152) VARYING INIT('');
DCL TIM_SER_INDX FIXED BIN;
DCL SIMDEFS RECORD SEQL INPUT;
DCL $FSTSIMDEFS BIT(1) INIT('1'B);
DCL ENDFILE$SIMDEF BIT(1) INIT('0'B);
DCL SIMDEF_S CHAR(10) VARYING INIT('');
DCL SIMDEF_INDX FIXED BIN;
DCL SIMDEF_INPUTREC_PTR PTR;
DCL SIMDEF_INPUTREC_S CHAR(10) VARYING INIT('');
DCL SIMDEF_INPUTREC_P CHAR(10) BASED (SIMDEF_INPUTREC_PTR);
DCL COEFFS RECORD SEQL INPUT;
DCL $FSTCOEFFS BIT(1) INIT('1'B);
DCL ENDFILE$COEFF BIT(1) INIT('0'B);
DCL COEFF_S CHAR(8) VARYING INIT('');
DCL COEFF_INDX FIXED BIN;
DCL COEFF_CO_REC_PTR PTR;
DCL COEFF_CO_REC_S CHAR(8) VARYING INIT('');
DCL COEFF_CO_REC_P CHAR(8) BASED (COEFF_CO_REC_PTR);
DCL TIM_SER_TS_REC_PTR PTR;
DCL TIM_SER_TS_REC_S CHAR(152) VARYING INIT('');
DCL TIM_SER_TS_REC_P CHAR(152) BASED (TIM_SER_TS_REC_PTR);
DCL TIM_SER_TS_REC_INDX FIXED BIN;
DCL SOLUTION_HDR_REC_PTR PTR;
DCL SOLUTION_HDR_REC_S CHAR(10) VARYING INIT('');
DCL SOLUTION_HDR_REC_P CHAR(10) BASED (SOLUTION_HDR_REC_PTR);
DCL $ITER_CNVRG_1 BIT(1) ;
DCL $ITER_CNTR_1 FIXED BIN ;
DCL SOLUTION_SOL_REC_PTR PTR;
DCL SOLUTION_SOL_REC_S CHAR(121) VARYING INIT('');
DCL SOLUTION_SOL_REC_P CHAR(121) BASED (SOLUTION_SOL_REC_PTR);
DCL SOLUTIONT RECORD SEQL OUTPUT;
DCL $FSTSOLUTIONT BIT(1) INIT('1'B);
DCL $ERROR_BUF CHAR(270) VAR;
DCL ERRORF FILE RECORD OUTPUT;
DCL ERRORF_BIT BIT(1) STATIC INIT('1'B);
DCL ($ERROR,$ACC_ERROR,$NOT_DONE)(20) BIT(1);
DCL $ERR_LAB(20) LABEL;
DCL $ERRSP$ FIXED BIN STATIC INITIAL (0);
DCL $TMP_VAL FLOAT BIN;
DCL $TMP_ERR BIT(1);
DECLARE
   1 COEFF,
    2 CO_REC(73),
     3 A DEC FLOAT(10);
DECLARE
   1 SIMDEF,
    2 INPUTREC,
     3 BEG_YR PIC'9999',
     3 PD_SIM PIC'999',
     3 LAG PIC'999';
```

```
DECLARE
   1 SOLUTION,
    2 SOL_GRP(2),
     3 HDR_REC,
      4 SM_PD_ID PIC'9999',
      4 SM_YR_ID PIC'BB9999',
     3 SOL_REC,
      4 II PIC'BB99.V(6)9',
      4 EX PIC'BB99.V(6)9',
      4 GOV PIC'BB99.V(6)9',
      4 IMSER PIC'BB99.V(6)9',
      4 IM01 PIC'BB99.V(6)9',
      4 IM24 PIC'BB99.V(6)9',
      4 IM3 PIC'BB99.V(6)9',
      4 CONS PIC'BB99.V(6)9',
      4 INV PIC'BB99.V(6)9',
      4 IM PIC'BB99.V(6)9',
      4 GDP PIC'BB99.V(6)9';
DECLARE
   1 TIM_SER,
    2 TS_REC(11),
     3 VNAME CHAR(4),
     3 VNUM PIC'9999',
     3 NUM_PDS PIC'9999',
     3 TS_DATA(20) PIC'S99.V999';
DECLARE
   1 INTERIM,
    2 SIZE$SOLUTION_SOL_GRP FIXED BIN ,
    2 $YSGEN1,
     3 SIZE$TIM_SER_TS_DATA(11) FIXED BIN ;
DCL F_E$COEFF_CO_REC FIXED BIN;
DCL F_E$SOLUTION_SOL_GRP FIXED BIN;
DCL F_E$TIM_SER_TS_REC FIXED BIN;
DCL F_E$TIM_SER_TS_DATA FIXED BIN;
DCL (TRUE,SELECTED) BIT(1) INIT('1'B);
DCL (FALSE,NOT_SELE,NOT_SELECTED) BIT(1) INIT('0'B);
ON ENDFILE(TIM_SERS) BEGIN;
ENDFILE$TIM_SER='1'B;
 TIM_SER_S=' ';
 END;
ON ENDFILE(SIMDEFS) BEGIN;
ENDFILE$SIMDEF='1'B;
 SIMDEF_S=' ';
 END;
ON ENDFILE(COEFFS) BEGIN;
ENDFILE$COEFF='1'B;
 COEFF_S=' ';
 END;
ON UNDEFINEDFILE(ERRORF)  ERRORF_BIT='0'B;
ERROR_RESTART: ;
$ERRSP$ = $ERRSP$ +1;
$ERROR($ERRSP$)='0'B;
$ACC_ERROR($ERRSP$)='0'B;
$ERR_LAB($ERRSP$)=END_PROGRAM;
```

```
OPEN FILE(TIM_SERS);
OPEN FILE(SIMDEFS);
IF $FSTSIMDEFS THEN DO;
 READ FILE(SIMDEFS) INTO (SIMDEF_INPUTREC_S);
 $FSTSIMDEFS='0'B;
 END;
ELSE SIMDEF_INPUTREC_S=SIMDEF_S;
SIMDEF_INPUTREC_PTR=ADDR(SIMDEF.INPUTREC);
SIMDEF_INPUTREC_P=SIMDEF_INPUTREC_S;
IF ^ENDFILE$SIMDEF THEN READ FILE(SIMDEFS) INTO (SIMDEF_S);
$ERROR_BUF=SIMDEF_INPUTREC_S;
OPEN FILE(COEFFS);
$ERRSP$ = $ERRSP$ +1;
$ERROR($ERRSP$)='0'B;
$ACC_ERROR($ERRSP$)='0'B;
$ERR_LAB($ERRSP$)=LOOP_END1;
DO F_E$COEFF_CO_REC =1 TO 73 WHILE(^ENDFILE$COEFF);
  $ERROR($ERRSP$)='0'B;
  IF $FSTCOEFFS THEN DO;
   READ FILE(COEFFS) INTO (COEFF_CO_REC_S);
   $FSTCOEFFS='0'B;
   END;
  ELSE COEFF_CO_REC_S=COEFF_S;
  COEFF_CO_REC_PTR=ADDR(COEFF.CO_REC(F_E$COEFF_CO_REC));
  COEFF_CO_REC_P=COEFF_CO_REC_S;
  IF ^ENDFILE$COEFF THEN READ FILE(COEFFS) INTO (COEFF_S);
  $ERROR_BUF=COEFF_CO_REC_S;
  LOOP_END1:;
END;
$TMP_ERR=$ACC_ERROR($ERRSP$);
$ERRSP$ = $ERRSP$ - 1;
IF $TMP_ERR THEN $ERROR($ERRSP$)='1'B;
IF $TMP_ERR THEN $ACC_ERROR($ERRSP$)='1'B;
SIZE$SOLUTION_SOL_GRP=SIMDEF.PD_SIM;
$ERRSP$ = $ERRSP$ +1;
$ERROR($ERRSP$)='0'B;
$ACC_ERROR($ERRSP$)='0'B;
$ERR_LAB($ERRSP$)=LOOP_END2;
DO F_E$TIM_SER_TS_REC =1 TO 11 WHILE(^ENDFILE$TIM_SER);
  $ERROR($ERRSP$)='0'B;
  IF $FSTTIM_SERS THEN DO;
   READ FILE(TIM_SERS) INTO (TIM_SER_TS_REC_S);
   $FSTTIM_SERS='0'B;
   END;
  ELSE TIM_SER_TS_REC_S=TIM_SER_S;
  TIM_SER_TS_REC_PTR=ADDR(TIM_SER.TS_REC(F_E$TIM_SER_TS_REC));
  TIM_SER_TS_REC_INDX=1;
  IF ^ENDFILE$TIM_SER THEN READ FILE(TIM_SERS) INTO (TIM_SER_S);
  $ERROR_BUF=TIM_SER_TS_REC_S;
  TIM_SER.VNAME(F_E$TIM_SER_TS_REC)=SUBSTR(TIM_SER_TS_REC_S,TIM_SER_TS_REC_INDX,
  4) ;
  TIM_SER_TS_REC_INDX=TIM_SER_TS_REC_INDX+4 ;
  TIM_SER.VNUM(F_E$TIM_SER_TS_REC)=SUBSTR(TIM_SER_TS_REC_S,TIM_SER_TS_REC_INDX,
  4) ;
  TIM_SER_TS_REC_INDX=TIM_SER_TS_REC_INDX+4 ;
  TIM_SER.NUM_PDS(F_E$TIM_SER_TS_REC)=SUBSTR(TIM_SER_TS_REC_S,
  TIM_SER_TS_REC_INDX,4) ;
  TIM_SER_TS_REC_INDX=TIM_SER_TS_REC_INDX+4 ;
  SIZE$TIM_SER_TS_DATA(F_E$TIM_SER_TS_REC)=TIM_SER.NUM_PDS(F_E$TIM_SER_TS_REC);
```

```
    $ERRSP$ = $ERRSP$ +1;
    $ERROR($ERRSP$)='0'B;
    $ACC_ERROR($ERRSP$)='0'B;
    $ERR_LAB($ERRSP$)=LOOP_END3;
    DO F_E$TIM_SER_TS_DATA =1 TO SIZE$TIM_SER_TS_DATA(F_E$TIM_SER_TS_REC);
      $ERROR($ERRSP$)='0'B;
      TIM_SER.TS_DATA(F_E$TIM_SER_TS_REC,F_E$TIM_SER_TS_DATA)=SUBSTR(
      TIM_SER_TS_REC_S,TIM_SER_TS_REC_INDX,7) ;
      TIM_SER_TS_REC_INDX=TIM_SER_TS_REC_INDX+7 ;
      LOOP_END3:;
    END;
    $TMP_ERR=$ACC_ERROR($ERRSP$);
    $ERRSP$ = $ERRSP$ - 1;
    IF $TMP_ERR THEN $ERROR($ERRSP$)='1'B;
    IF $TMP_ERR THEN $ACC_ERROR($ERRSP$)='1'B;
    LOOP_END2:;
  END;
  $TMP_ERR=$ACC_ERROR($ERRSP$);
  $ERRSP$ = $ERRSP$ - 1;
  IF $TMP_ERR THEN $ERROR($ERRSP$)='1'B;
  IF $TMP_ERR THEN $ACC_ERROR($ERRSP$)='1'B;
  $ERRSP$ = $ERRSP$ +1;
  $ERROR($ERRSP$)='0'B;
  $ACC_ERROR($ERRSP$)='0'B;
  $ERR_LAB($ERRSP$)=LOOP_END4;
  DO F_E$SOLUTION_SOL_GRP =1 TO SIZE$SOLUTION_SOL_GRP;
    $ERROR($ERRSP$)='0'B;
    SOLUTION.SM_PD_ID(2)=F_E$SOLUTION_SOL_GRP;
    SOLUTION.SM_YR_ID(2)=SIMDEF.BEG_YR+F_E$SOLUTION_SOL_GRP-1;
    SOLUTION_HDR_REC_PTR=ADDR(SOLUTION.HDR_REC(2));
    SOLUTION_HDR_REC_S = SOLUTION_HDR_REC_P;
    WRITE FILE(SOLUTIONT) FROM (SOLUTION_HDR_REC_S);
    SOLUTION_HDR_REC_S='';
    SOLUTION.EX(2)=TIM_SER.TS_DATA(4,F_E$SOLUTION_SOL_GRP);
    SOLUTION.GOV(2)=TIM_SER.TS_DATA(6,F_E$SOLUTION_SOL_GRP);
    SOLUTION.IMSER(2)=TIM_SER.TS_DATA(8,F_E$SOLUTION_SOL_GRP);
    SOLUTION.IM01(2)=TIM_SER.TS_DATA(9,F_E$SOLUTION_SOL_GRP);
    SOLUTION.IM24(2)=TIM_SER.TS_DATA(10,F_E$SOLUTION_SOL_GRP);
    SOLUTION.IM3(2)=TIM_SER.TS_DATA(11,F_E$SOLUTION_SOL_GRP);
    SOLUTION.II(2)=TIM_SER.TS_DATA(3,F_E$SOLUTION_SOL_GRP);
    IF (F_E$SOLUTION_SOL_GRP > 1) THEN SOLUTION.GDP(2) = SOLUTION.GDP(1);
    ELSE SOLUTION.GDP(2) = 1;
    IF (F_E$SOLUTION_SOL_GRP > 1) THEN SOLUTION.IM(2) = SOLUTION.IM(1);
    ELSE SOLUTION.IM(2) = 1;
    IF (F_E$SOLUTION_SOL_GRP > 1) THEN SOLUTION.INV(2) = SOLUTION.INV(1);
    ELSE SOLUTION.INV(2) = 1;
    IF (F_E$SOLUTION_SOL_GRP > 1) THEN SOLUTION.CONS(2) = SOLUTION.CONS(1);
    ELSE SOLUTION.CONS(2) = 1;
```

```
  $ITER_PROC_$MAINBLOCK: PROCEDURE($OLD_VAL,$NEW_VAL,$VAR_NAME);
    DCL ($OLD_VAL,$NEW_VAL,$REL_ERR) FLOAT BIN;
    DCL $VAR_NAME CHAR(32) VAR;
    IF $OLD_VAL = 0.0 THEN $REL_ERR = ABS($NEW_VAL);
    ELSE $REL_ERR = ABS($OLD_VAL-$NEW_VAL)/ABS($OLD_VAL);
    IF $REL_ERR >  1.000000E-04 THEN DO;
      IF $ITER_CNTR_1 = 100 THEN DO;
        $ERROR_BUF = 'NO CONVERGENCE IN BLOCK: $MAINBLOCK';
        WRITE FILE (ERRORF) FROM ($ERROR_BUF);
        $ERROR_BUF = '  VARIABLE NAME :  '|| $VAR_NAME;
        WRITE FILE (ERRORF) FROM ($ERROR_BUF);
        $ERROR_BUF = '  CURRENT VALUE : '|| $NEW_VAL;
        WRITE FILE (ERRORF) FROM ($ERROR_BUF);
        $ERROR_BUF = '  PREVIOUS VALUE: '|| $OLD_VAL;
        WRITE FILE (ERRORF) FROM ($ERROR_BUF);
        $ERROR_BUF = '  RELATIVE ERROR: '|| $REL_ERR;
        WRITE FILE (ERRORF) FROM ($ERROR_BUF);
      END;
      $ITER_CNVRG_1 = '0'B;
    END;
    RETURN;
  END $ITER_PROC_$MAINBLOCK;
  $ITER_CNVRG_1 = '0'B;
  DO $ITER_CNTR_1 = 1 TO 100 WHILE (^$ITER_CNVRG_1);
    $ITER_CNVRG_1= '1'B;
    $TMP_VAL = SOLUTION.CONS(2);
    IF F_E$SOLUTION_SOL_GRP>SIMDEF.LAG THEN SOLUTION.CONS(2) = COEFF.A(1)
    +COEFF.A(2)*SOLUTION.GDP(2)+COEFF.A(3)*SOLUTION.CONS(1);
     ELSE SOLUTION.CONS(2) = TIM_SER.TS_DATA(1,F_E$SOLUTION_SOL_GRP);
    CALL $ITER_PROC_$MAINBLOCK($TMP_VAL,SOLUTION.CONS(2),'SOLUTION.CONS(2)');
    $TMP_VAL = SOLUTION.INV(2);
    IF F_E$SOLUTION_SOL_GRP>SIMDEF.LAG THEN SOLUTION.INV(2) = COEFF.A(4)
    +COEFF.A(5)*SOLUTION.GDP(2)+COEFF.A(6)*SOLUTION.GDP(1)+SOLUTION.II(2);
     ELSE SOLUTION.INV(2) = TIM_SER.TS_DATA(2,F_E$SOLUTION_SOL_GRP);
    CALL $ITER_PROC_$MAINBLOCK($TMP_VAL,SOLUTION.INV(2),'SOLUTION.INV(2)');
    $TMP_VAL = SOLUTION.IM(2);
    IF F_E$SOLUTION_SOL_GRP>SIMDEF.LAG THEN SOLUTION.IM(2) = SOLUTION.IM01(2)
    +SOLUTION.IM24(2)+SOLUTION.IM3(2)+COEFF.A(61)+COEFF.A(62)*SOLUTION.GDP(2)
    +SOLUTION.IMSER(2);
     ELSE SOLUTION.IM(2) = TIM_SER.TS_DATA(5,F_E$SOLUTION_SOL_GRP);
    CALL $ITER_PROC_$MAINBLOCK($TMP_VAL,SOLUTION.IM(2),'SOLUTION.IM(2)');
    $TMP_VAL = SOLUTION.GDP(2);
    IF F_E$SOLUTION_SOL_GRP>SIMDEF.LAG THEN SOLUTION.GDP(2) = SOLUTION.CONS(2)
    +SOLUTION.INV(2)+SOLUTION.EX(2)+SOLUTION.GOV(2)-SOLUTION.IM(2);
     ELSE SOLUTION.GDP(2) = TIM_SER.TS_DATA(7,F_E$SOLUTION_SOL_GRP);
    CALL $ITER_PROC_$MAINBLOCK($TMP_VAL,SOLUTION.GDP(2),'SOLUTION.GDP(2)');
  END /* $ITER_CNTR_1 */ ;
  SOLUTION_SOL_REC_PTR=ADDR(SOLUTION.SOL_REC(2));
  SOLUTION_SOL_REC_S = SOLUTION_SOL_REC_P;
  WRITE FILE(SOLUTIONT) FROM (SOLUTION_SOL_REC_S);
  SOLUTION_SOL_REC_S='';
  SOLUTION.SOL_GRP(1) = SOLUTION.SOL_GRP(2);
  LOOP_END4:;
END;
$TMP_ERR=$ACC_ERROR($ERRSP$);
$ERRSP$ = $ERRSP$ - 1;
IF $TMP_ERR THEN $ERROR($ERRSP$)='1'B;
IF $TMP_ERR THEN $ACC_ERROR($ERRSP$)='1'B;
CLOSE FILE(SOLUTIONT);
END_PROGRAM: RETURN;
END SPAIN;
```

APPENDIX III

PROGRAM LISTING
OF MODIFICATIONS TO
THE MODEL SCHEDULER

```
simul_blk: proc returns(ptr);
     dcl pair_ptr ptr;
     dcl 1 pair based (pair_ptr),
      2 cdr ptr,
      2 car fixed bin;
     dcl gnode_ptr ptr;
     dcl 1 gnode based (gnode_ptr),
      2 nxt_gnode ptr,
      2 node_id fixed bin,
      2 suxl ptr;
     dcl ppair_ptr ptr;
     dcl 1 ppair based (ppair_ptr),
      2 pcdr ptr,
      2 pcar ptr;
     dcl edge_ptr ptr;
     dcl 1 edge based (edge_ptr),
      2 source fixed bin,
      2 target fixed bin,
      2 edge_type fixed bin,
      2 dim_dif fixed bin,
      2 subx ptr;
    dcl succ_list entry(fixed bin) returns (ptr);
    dcl sim_ptr ptr;              /* pointer to current sim structure   */
    dcl 1 sim based (sim_ptr),    /* structure used to order graph      */
      2 sim_node_id fixed bin,    /* node id of current graph node      */
      2 sim_target fixed bin,     /* target node of type 7 ASTX edge    */
      2 sim_stmt$ fixed bin,      /* stmt # of current graph node       */
      2 sim_gnode ptr,            /* ptr to final gnode of this sim     */
      2 nxt_sim ptr;              /* ptr to next sim structure          */
    dcl bp pointer;               /* pointer to block structure         */
    dcl 1 block based (bp),       /* structure of block information     */
     2 btype char(4),
     2 bb_stmt$ fixed bin,        /* beginning stmt# of this block      */
     2 be_stmt$ fixed bin,        /* ending stmt# of this block         */
     2 blabel char(max_len_name), /* label of this block                */
     2 blevel fixed bin,          /* nesting level of this block        */
     2 bmethod fixed bin,         /* desired solution method            */
     2 bmax_iter float dec,       /* maximum number of iterations       */
     2 brel_error float dec,      /* solution tolerance specification   */
     2 bparent ptr,               /* ptr to parent block structure      */
     2 bup ptr,                   /* ptr to previous block structure    */
     2 bdown ptr;                 /* ptr to next block structure        */
    dcl (hol,eol) ptr;            /* head-of-list,end-of-list for sim   */
    dcl bhol ptr ext;             /* head-of-list for block structures  */
    dcl first_stmt$ fixed bin     /* stmt# of first assertion in mscc   */
        initial(999999);          /* # higher than any real ass. #      */
    dcl last_stmt$  fixed bin     /* stmt#  of last  assertion in mscc  */
        initial(0);               /* # lower than any real ass. #       */
    dcl temp_ptr ptr;             /* temporary ptr for list manip.      */
  put skip list ('starting scheduling of simultaneous block');
/*********************************************************************/
/* initialize the list of sim structures with hol and eol members    */
/*********************************************************************/
  allocate sim set(hol);          /* allocate space for head-of-list    */
  allocate sim set(eol);          /* allocate space for end-of-list     */
  hol -> sim_stmt$ = 0;           /* 0 is lower than any real stmt #    */
  eol -> sim_stmt$ = 999999;      /* # is higher than any real stmt #   */
  hol -> nxt_sim = eol;           /* list starts empty                  */
  eol -> nxt_sim = null;          /* end-of-list has no next member     */
```

```
/*************************************************************************/
/* construct a sim data structure for each ASTX in input-file order      */
/*************************************************************************/
  gnode_ptr = graph;              /* start at the first subgraph node   */
  do while (gnode_ptr ^= null);   /* look through all graph nodes       */
   if(dictype(node_id)='ASTX') then do;
     allocate sim;                /* create a list entry for assertion  */
     sim_node_id = node_id;       /* node id of current graph node      */
     sim_gnode = gnode_ptr;       /* ptr to this node of the graph      */
     sim_stmt$ = bin(substr(dict(node_id),5));
     if (sim_stmt$<first_stmt$) then first_stmt$=sim_stmt$;
     if (sim_stmt$>last_stmt$ ) then last_stmt$ =sim_stmt$;
     ppair_ptr = succ_list(node_id);  /* find target node of type 7 edge */
     edge_ptr = pcar;
     do while (edge_type^=7);
      ppair_ptr = pcdr;
      edge_ptr = pcar;
     end;
     sim_target = target;             /* target edge of this assertion   */
     temp_ptr = hol;                  /* now, find place to insert stmt. */
     do while (temp_ptr->nxt_sim->sim_stmt$ < sim_stmt$);
        temp_ptr = temp_ptr -> nxt_sim;
     end;
     nxt_sim = temp_ptr -> nxt_sim;   /* now,insert sim into list        */
     temp_ptr -> nxt_sim = sim_ptr;
   end;
   gnode_ptr = nxt_gnode;             /* advance ptr to next graph node  */
  end;
/*************************************************************************/
/* construct a sim data structure for each FLD target of the             */
/* preceding ASTX elements                                               */
/*************************************************************************/
  gnode_ptr = graph;            /*  start at subgraph beginning         */
  do while(gnode_ptr ^= null);
    if (dictype(node_id)='FLD') then do;
      temp_ptr = hol->nxt_sim;  /* now insert entry into after its ass. */
      do while(temp_ptr ^= eol);
        if (temp_ptr->sim_target = node_id) then do;
          allocate sim;             /* create a list entry for field node   */
          sim_node_id = node_id;
          sim_gnode = gnode_ptr;
          sim_stmt$ = temp_ptr->sim_stmt$;
          nxt_sim = temp_ptr -> nxt_sim;
          temp_ptr -> nxt_sim = sim_ptr;
        end;
        temp_ptr = temp_ptr -> nxt_sim;
      end;
    end;
    gnode_ptr = nxt_gnode;
  end;
```

```
/*********************************************************/
/*  find the last sim structure in the first component    */
/*********************************************************/
dcl after ptr;      /* will point to sim after last in first component */
dcl first_bp ptr;   /* bp of the first component in this mscc          */
first_bp = null;
bp = bhol;
do while ((bp^=null) & (first_bp=null));
  if ((bb_stmt$<first_stmt$) & (be_stmt$>first_stmt$) &
      (be_stmt$<last_stmt$))  then first_bp = bp;
  bp = bdown;
end;
sim_ptr = hol;
if (first_bp=null) then sim_ptr = nxt_sim->nxt_sim;
else do while (nxt_sim->sim_stmt$ < first_bp->be_stmt$);
  sim_ptr = nxt_sim;
end;
after = nxt_sim;
put skip list('first_bp,after',first_bp->blabel,dict(after->sim_node_id));
/*************************************************************/
/* make a list of node_id's in the first component           */
/*************************************************************/
dcl del_list ptr;
dcl del_ptr ptr;
dcl 1 del_node based (del_ptr),
 2 del_node_id fixed bin,
 2 nxt_del ptr;
del_list = null;
sim_ptr = hol->nxt_sim;
do while (sim_ptr^=after);
  allocate del_node;
  del_node_id = sim_node_id;
  nxt_del = del_list;
  del_list = del_ptr;
  sim_ptr = nxt_sim;
end;
/*************************************************************/
/*  print out graph edges for debugging                      */
/*************************************************************/
if (trace) then do;
  put skip list ('printing graph edges');
  sim_ptr = hol->nxt_sim;
  do while(sim_ptr^=eol);
    gnode_ptr = sim_gnode;
    if (suxl=null) then put skip list ('no edges from ',dict(node_id),node_id);
    else do;
      temp_ptr = suxl;
      do while(temp_ptr^=null);
        edge_ptr=temp_ptr->pcar;
        put skip list ('edge:',dict(source),dict(target),source,target);
        temp_ptr=temp_ptr->pcdr;
      end;
    end;
    sim_ptr=nxt_sim;
   end;
end;
```

```
/*************************************************************/
/*  delete all backward edges into the first component     */
/*************************************************************/
dcl prec ptr;         /* points to preceding ppair */
dcl found bit;
sim_ptr = after;
do while (sim_ptr^=eol);
  ppair_ptr = sim_gnode->suxl;
  prec = ppair_ptr;
  do while (ppair_ptr^=null);
    edge_ptr = pcar;
    del_ptr = del_list;
    found = '0'b;
    do while (del_ptr^=null & ^found);
      if (target = del_node_id) then found='1'b;
      del_ptr = nxt_del;
    end;
    if (found) then do;
      put skip list ('removing edge',dict(source),dict(target));
      if (sim_gnode->suxl=ppair_ptr) then sim_gnode->suxl=pcdr;
      else prec->pcdr = pcdr;
    end;
    else prec = ppair_ptr;
    ppair_ptr = pcdr;
  end;
  sim_ptr = nxt_sim;
end;
/*********************************************************************/
/* find the smallest block which contains our entire mscc            */
/*********************************************************************/
  dcl outerblock ptr;         /* ptr to smallest containing block of mscc */
  bp = bhol;
  do while (bp^=null);
    if ((bb_stmt$<first_stmt$) & (be_stmt$>last_stmt$)) then outerblock=bp;
    bp = bdown;
  end;
  put skip list ('outerblock is',outerblock->blabel);
```

```
/*************************************************************************/
/* allocate block if necessary and recursively schedule or terminate */
/*************************************************************************/
bp = outerblock;
if (btype^='DONE') then do;
  btype = 'DONE';
  allocate selmnt set (slmn_ptr);
  nxt_slmn = null;
  slmn_type = 3;
  slmn_label = blabel;
  put skip list ('allocating block ',blabel);
  slmn_method = bmethod;
  slmn_max_iter = bmax_iter;
  slmn_rel_error = brel_error;
  slmn_name = block_level;
  slmn_vars = null;
  past_names(level) = slmn_name;
  if (block_level = 1) then do;
    sim_ptr = hol -> nxt_sim;
    do while(sim_ptr ^=eol);          /* create var list for initial. */
      if(dictype(sim_node_id)='FLD') then do;
        allocate pair set (pair_ptr);
        car = sim_node_id;
        cdr = slmn_vars;
        slmn_vars = pair_ptr;
      end;
      sim_ptr = nxt_sim;
    end;
  end;
  slmn_list = schedule_graph(graph,level+1,block_level+1);
  btype = '    ';
  sim_ptr = hol;               /* free sim structures */
  do while(sim_ptr^=null);
    temp_ptr = nxt_sim;
    free sim;
    sim_ptr = temp_ptr;
  end;
  del_ptr = del_list;          /* free del list */
  do while (del_ptr^=null);
    temp_ptr = nxt_del;
    free del_node;
    del_ptr = temp_ptr;
  end;
  return(slmn_ptr);
end;
else return(schedule_graph(graph,level,block_level));
 end simul_blk;
```

APPENDIX IV

MODEL SPECIFICATION

OF THE

MODEL LINKED WORLD TRADE MODEL

```
MODULE: LINK;              /*  LINKED WORLD TRADE MODEL                */

/* THIS IS A LINKAGE MODEL BASED ON THE METHOD OF CONSTANT VALUE SHARES.    */
/* THERE ARE FOUR SECTIONS IN THIS MODULAR PRESENTATION, NAMELY             */
/*     1) ROW - A SIMPLE EXOGENOUS SOLUTION FOR MOST OF THE WORLD           */
/*     2) LINK - A LINKAGE SECTION BASED ON THE CVS METHOD                  */
/*     3) FRANCE - THE POM-POM MODEL OF THE FRENCH ECONOMY                  */
/*     4) SPAIN  - A MODEL OF THE SPANISH ECONOMY                           */
/* THE MODELS ARE PRESENTED IN THIS ORDER. TOGETHER THEY COMPRISE THE TOTAL */
/* MODEL LANGUAGE INPUT SPECIFICATION FOR THE WORLD TRADE MODEL.            */

/* THE FOLLOWING CONVENTIONS ARE USED IN THE NAMING OF THE LINKAGE VARIABLES */
/*                                                                          */
/*         X       -   EXPORTS                                              */
/*         M       -   IMPORTS                                              */
/*         F       -   FRANCE (FRENCH)                                      */
/*         S       -   SPAIN  (SPANISH)                                     */
/*         R       -   ROW    (REST OF WORLD)                               */
/*         TW      -   TOTAL WORLD                                          */
/*         P       -   PRICE                                                */
/*         V       -   VALUE                                                */
/*         DIGITS  -   COMMODITY GROUPS (01,24,3,59,09)                     */

SOURCE FILE: SIMDEF;     /* PARAMETER SPECIFICATION FOR SIMULATION  */

SIMDEF IS FILE (INPTREC);          /* SIMULATION DEFINITION FILE SPECIFIES */
                                   /* CONTROLLING PARAMETERS FOR THE SIM.  */

  INPTREC IS RECORD(BEG_YR,PD_SIM,LAG);

    BEG_YR   IS FIELD (PIC'9999');  /* YEAR OF START OF SIMULATION PRINTOUT  */
    PD_SIM   IS FIELD (PIC'999');   /* NUMBER OF PERIODS IN OUR SIMULATION   */
    LAG      IS FIELD (PIC'999');   /* NUMBER OF PDS.  OF SIMULATION BEFORE  */
                                    /*  USING MODEL EQUATIONS                */
```

```
/*********************************************************************/
/*                                                                   */
/*                        LINKAGE MODEL                              */
/*                                                                   */
/*********************************************************************/

TARGET FILE: LKS;         /* SIMULATION RESULTS FOR ALL CVS VARS      */

LKS IS FILE (LKGROUP);            /* LINK SOLUTION FILE CONTAINS VALUES OF  */
                                  /*  IMPORT-EXPORT LINKAGE VARIABLES       */
LKGROUP IS GROUP(LKHDR,T1,LKP1,T2,LKP2,T3,LKP3,T4,LKP4,T5,LKP5,T6,LKP6,T7,LKP7);

  LKHDR IS RECORD(TITLE);                       TITLE IS FIELD (CHAR(200));
  LKS.TITLE = 'LINKAGE VARS: JOINT LINK/MODEL ECONOMETRIC RESEARCH GROUP';
  T1 IS RECORD(LAB1(2)); T2 IS RECORD(LAB2(2)); T3 IS RECORD(LAB3(2));
  T4 IS RECORD(LAB4(2)); T5 IS RECORD(LAB5(2)); T6 IS RECORD(LAB6(2));
  T7 IS RECORD(LAB7(2));

  (LAB1,LAB2,LAB3,LAB4,LAB5,LAB6,LAB7) ARE FIELD (CHAR(200));

 (LKS.LAB1(1),LKS.LAB2(1),LKS.LAB3(1),LKS.LAB4(1),
  LKS.LAB5(1),LKS.LAB6(1),LKS.LAB7(1))  = ' ';

LKS.LAB1(2) = '       VXF01       VXF24       VXF3        VXF59       '||
        'VXF09       VXS01       VXS24       VXS3        VXS59       '||
        'VXS09       VXR01       VXR24       VXR3        VXR59       '||
        'VXR09';
LKS.LAB2(2) ='        MF01        MF24        MF3         MF59        '||
        'MF09        MS01        MS24        MS3         MS59        '||
        'MS09        MR01        MR24        MR3         MR59        '||
        'MR09';
LKS.LAB3(2) ='       PMF01       PMF24       PMF3        PMF59       '||
        'PMF09       PMS01       PMS24       PMS3        PMS59       '||
        'PMS09       PMR01       PMR24       PMR3        PMR59       '||
        'PMR09';
LKS.LAB4(2) ='       VMF01       VMF24       VMF3        VMF59       '||
        'VMF09       VMS01       VMS24       VMS3        VMS59       '||
        'VMS09       VMR01       VMR24       VMR3        VMR59       '||
        'VMR09';
LKS.LAB5(2) ='       PXF01       PXF24       PXF3        PXF59       '||
        'PXF09       PXS01       PXS24       PXS3        PXS59       '||
        'PXS09       PXR01       PXR24       PXR3        PXR59       '||
        'PXR09';
LKS.LAB6(2) ='        XF01        XF24        XF3         XF59        '||
        'XF09        XS01        XS24        XS3         XS59        '||
        'XS09        XR01        XR24        XR3         XR59        '||
        'XR09';
LKS.LAB7(2) ='       TWVX01      TWVX24      TWVX3       TWVX59      '||
        'TWVX09      TWX01       TWX24       TWX3        TWX59       '||
        'TWX09       TWPX01      TWPX24      TWPX3       TWPX59      '||
        'TWPX09';

  LKP1 IS GROUP (LKREC1(1:9));  LKP2 IS GROUP (LKREC2(1:9));
  LKP3 IS GROUP (LKREC3(1:9));  LKP4 IS GROUP (LKREC4(1:9));
  LKP5 IS GROUP (LKREC5(1:9));  LKP6 IS GROUP (LKREC6(1:9));
  LKP7 IS GROUP (LKREC7(1:9));

 (SIZE.LKREC1,SIZE.LKREC2,SIZE.LKREC3,SIZE.LKREC4,
  SIZE.LKREC5,SIZE.LKREC6,SIZE.LKREC7)    =    PD_SIM;
```

```
  LKREC1 IS RECORD(YR1,VXF01,VXF24,VXF3,VXF59,VXF09,VXS01,VXS24,VXS3,VXS59,
                   VXS09,VXR01,VXR24,VXR3,VXR59,VXR09);
  LKREC2 IS RECORD(YR2,MF01,MF24,MF3,MF59,MF09,MS01,MS24,MS3,MS59,MS09,
                   MR01,MR24,MR3,MR59,MR09);
  LKREC3 IS RECORD(YR3,PMF01,PMF24,PMF3,PMF59,PMF09,PMS01,PMS24,PMS3,PMS59,
                   PMS09,PMR01,PMR24,PMR3,PMR59,PMR09);
  LKREC4 IS RECORD(YR4,VMF01,VMF24,VMF3,VMF59,VMF09,VMS01,VMS24,VMS3,VMS59,
                   VMS09,VMR01,VMR24,VMR3,VMR59,VMR09);
  LKREC5 IS RECORD(YR5,PXF01,PXF24,PXF3,PXF59,PXF09,PXS01,PXS24,PXS3,PXS59,
                   PXS09,PXR01,PXR24,PXR3,PXR59,PXR09);
  LKREC6 IS RECORD(YR6,XF01,XF24,XF3,XF59,XF09,XS01,XS24,XS3,XS59,XS09,
                      XR01,XR24,XR3,XR59,XR09);
  LKREC7 IS RECORD(YR7,TWVX01,TWVX24,TWVX3,TWVX59,TWVX09,TWX01,TWX24,TWX3,
                   TWX59,TWX09,TWPX01,TWPX24,TWPX3,TWPX59,TWPX09);

 (YR1,YR2,YR3,YR4,YR5,YR6,YR7,YR8)  ARE FIELD (PIC'9999');

 (LKS.YR1(T),LKS.YR2(T),LKS.YR3(T),LKS.YR4(T),
  LKS.YR5(T),LKS.YR6(T),LKS.YR7(T))        =  BEG_YR + T - 1;

(VXFS01,VXFR01,VXSF01,VXSR01,VXRS01,VXRF01,VXRR01,VXFS24,VXFR24,VXSF24,
 VXSR24,VXRS24,VXRF24,VXRR24,VXFS3 ,VXFR3 ,VXSF3 ,VXSR3 ,VXRS3 ,VXRF3 ,
 VXRR3 ,VXFS59,VXFR59,VXSF59,VXSR59,VXRS59,VXRF59,VXRR59,VXFS09,VXFR09,
 VXSF09,VXSR09,VXRS09,VXRF09,VXRR09,VXF01 ,VXF24 ,VXF3  ,VXF59 ,VXF09 ,
 VXS01 ,VXS24 ,VXS3  ,VXS59 ,VXS09 ,VXR01 ,VXR24 ,VXR3  ,VXR59 ,VXR09 ,
 TWVX01,TWVX24,TWVX3 ,TWVX59,TWVX09,MF01  ,MF24  ,MF3   ,MF59  ,MF09  ,
 MS01  ,MS24  ,MS3   ,MS59  ,MS09  ,MR01  ,MR24  ,MR3   ,MR59  ,MR09  ,
 PMF01 ,PMF24 ,PMF3  ,PMF59 ,PMF09 ,PMS01 ,PMS24 ,PMS3  ,PMS59 ,PMS09 ,
 PMR01 ,PMR24 ,PMR3  ,PMR59 ,PMR09 ,XF01  ,XF24  ,XF3   ,XF59  ,XF09  ,
 XS01  ,XS24  ,XS3   ,XS59  ,XS09  ,XR01  ,XR24  ,XR3   ,XR59  ,XR09  ,
 TWX01 ,TWX24 ,TWX3  ,TWX59 ,TWX09 ,TWPX01,TWPX24,TWPX3 ,TWPX59,TWPX09)

             ARE FIELD (PIC'BB(5)-9V.(4)9');

T IS SUBSCRIPT;        /* T (TIME) IS A COUNTER OF SIMULATION PDS. */

BLOCK LINKMOD: MAX ITER IS 10, RELATIVE ERROR IS 0.00025;

VXFS01(T) = 0.08100 * VMS01(T);
VXFR01(T) = 0.07380 * VMR01(T);
VXSF01(T) = 0.03870 * VMF01(T);
VXSR01(T) = 0.01337 * VMR01(T);
VXRS01(T) = 0.91900 * VMS01(T);
VXRF01(T) = 0.96130 * VMF01(T);
VXRR01(T) = 0.91283 * VMR01(T);

VXFS24(T) = 0.07200 * VMS24(T);
VXFR24(T) = 0.03340 * VMR24(T);
VXSF24(T) = 0.01000 * VMF24(T);
VXSR24(T) = 0.00614 * VMR24(T);
VXRS24(T) = 0.92800 * VMS24(T);
VXRF24(T) = 0.99000 * VMF24(T);
VXRR24(T) = 0.96046 * VMR24(T);
```

```
VXFS3 (T) = 0.01560 * VMS3 (T);
VXFR3 (T) = 0.00929 * VMR3 (T);
VXSF3 (T) = 0.00025 * VMF3 (T);
VXSR3 (T) = 0.00166 * VMR3 (T);
VXRS3 (T) = 0.98440 * VMS3 (T);
VXRF3 (T) = 0.99975 * VMF3 (T);
VXRR3 (T) = 0.98906 * VMR3 (T);

VXFS59(T) = 0.17360 * VMS59(T);
VXFR59(T) = 0.07490 * VMR59(T);
VXSF59(T) = 0.03000 * VMF59(T);
VXSR59(T) = 0.00948 * VMR59(T);
VXRS59(T) = 0.82640 * VMS59(T);
VXRF59(T) = 0.97000 * VMF59(T);
VXRR59(T) = 0.91562 * VMR59(T);

VXFS09(T) = VXFS01(T)+VXFS24(T)+VXFS3(T)+VXFS59(T);
VXFR09(T) = VXFR01(T)+VXFR24(T)+VXFR3(T)+VXFR59(T);
VXSF09(T) = VXSF01(T)+VXSF24(T)+VXSF3(T)+VXSF59(T);
VXSR09(T) = VXSR01(T)+VXSR24(T)+VXSR3(T)+VXSR59(T);
VXRS09(T) = VXRS01(T)+VXRS24(T)+VXRS3(T)+VXRS59(T);
VXRF09(T) = VXRF01(T)+VXRF24(T)+VXRF3(T)+VXRF59(T);
VXRR09(T) = VXRR01(T)+VXRR24(T)+VXRR3(T)+VXRR59(T);

VXF01(T) = VXFS01(T)+VXFR01(T);
VXF24(T) = VXFS24(T)+VXFR24(T);
VXF3 (T) = VXFS3 (T)+VXFR3 (T);
VXF59(T) = VXFS59(T)+VXFR59(T);
VXF09(T) = VXF01(T)+VXF24(T)+VXF3(T)+VXF59(T);

VXS01(T) = VXSF01(T)+VXSR01(T);
VXS24(T) = VXSF24(T)+VXSR24(T);
VXS3 (T) = VXSF3 (T)+VXSR3 (T);
VXS59(T) = VXSF59(T)+VXSR59(T);
VXS09(T) = VXS01(T)+VXS24(T)+VXS3(T)+VXS59(T);

VXR01(T) = VXRF01(T)+VXRS01(T)+VXRR01(T) ;
VXR24(T) = VXRF24(T)+VXRS24(T)+VXRR24(T) ;
VXR3 (T) = VXRF3 (T)+VXRS3 (T)+VXRR3 (T) ;
VXR59(T) = VXRF59(T)+VXRS59(T)+VXRR59(T) ;
VXR09(T) = VXR01(T)+VXR24(T)+VXR3(T)+VXR59(T);

TWVX01(T) = VXF01(T)+VXS01(T)+VXR01(T);
TWVX24(T) = VXF24(T)+VXS24(T)+VXR24(T);
TWVX3 (T) = VXF3 (T)+VXS3 (T)+VXR3 (T);
TWVX59(T) = VXF59(T)+VXS59(T)+VXR59(T);
TWVX09(T) = TWVX01(T)+TWVX24(T)+TWVX3(T)+TWVX59(T);

MF01(T) = VXSF01(T)/PXS01(T)+VXRF01(T)/PXR01(T);
MF24(T) = VXSF24(T)/PXS24(T)+VXRF24(T)/PXR24(T);
MF3 (T) = VXSF3 (T)/PXS3 (T)+VXRF3 (T)/PXR3 (T);
MF59(T) = VXSF59(T)/PXS59(T)+VXRF59(T)/PXR59(T);
MF09(T) = MF01(T)+MF24(T)+MF3(T)+MF59(T);

MS01(T) = VXFS01(T)/PXF01(T)+VXRS01(T)/PXR01(T);
MS24(T) = VXFS24(T)/PXF24(T)+VXRS24(T)/PXR24(T);
MS3 (T) = VXFS3 (T)/PXF3 (T)+VXRS3 (T)/PXR3 (T);
MS59(T) = VXFS59(T)/PXF59(T)+VXRS59(T)/PXR59(T);
MS09(T) = MS01(T)+MS24(T)+MS3(T)+MS59(T);
```

```
MR01(T) =
         VXFR01(T)/PXF01(T) + VXSR01(T)/PXS01(T) + VXRR01(T)/PXR01(T);
MR24(T) =
         VXFR24(T)/PXF24(T) + VXSR24(T)/PXS24(T) + VXRR24(T)/PXR24(T);
MR3 (T) =
         VXFR3 (T)/PXF3 (T) + VXSR3 (T)/PXS3 (T) + VXRR3 (T)/PXR3 (T);
MR59(T) =
         VXFR59(T)/PXF59(T) + VXSR59(T)/PXS59(T) + VXRR59(T)/PXR59(T);
MR09(T) = MR01(T)+MR24(T)+MR3(T)+MR59(T);

PMF01(T) = (VXSF01(T)+VXRF01(T))/MF01(T);
PMF24(T) = (VXSF24(T)+VXRF24(T))/MF24(T);
PMF3 (T) = (VXSF3 (T)+VXRF3 (T))/MF3 (T);
PMF59(T) = (VXSF59(T)+VXRF59(T))/MF59(T);
PMF09(T) = (VXSF09(T)+VXRF09(T))/MF09(T);

PMS01(T) = (VXFS01(T)+VXRS01(T))/MS01(T);
PMS24(T) = (VXFS24(T)+VXRS24(T))/MS24(T);
PMS3 (T) = (VXFS3 (T)+VXRS3 (T))/MS3 (T);
PMS59(T) = (VXFS59(T)+VXRS59(T))/MS59(T);
PMS09(T) = (VXFS09(T)+VXRS09(T))/MS09(T);

PMR01(T) = (VXFR01(T)+VXSR01(T)+VXRR01(T))/MR01(T);
PMR24(T) = (VXFR24(T)+VXSR24(T)+VXRR24(T))/MR24(T);
PMR3 (T) = (VXFR3 (T)+VXSR3 (T)+VXRR3 (T))/MR3 (T);
PMR59(T) = (VXFR59(T)+VXSR59(T)+VXRR59(T))/MR59(T);
PMR09(T) = (VXFR09(T)+VXSR09(T)+VXRR09(T))/MR09(T);

XF01 (T) = VXF01(T)/PXF01(T);
XF24 (T) = VXF24(T)/PXF24(T);
XF3  (T) = VXF3 (T)/PXF3 (T);
XF59 (T) = VXF59(T)/PXF59(T);
XF09 (T) = XF01(T)+XF24(T)+XF3(T)+XF59(T);

XS01 (T) = VXS01(T)/PXS01(T);
XS24 (T) = VXS24(T)/PXS24(T);
XS3  (T) = VXS3 (T)/PXS3 (T);
XS59 (T) = VXS59(T)/PXS59(T);
XS09 (T) = XS01(T)+XS24(T)+XS3(T)+XS59(T);

XR01 (T) = VXR01(T)/PXR01(T);
XR24 (T) = VXR24(T)/PXR24(T);
XR3  (T) = VXR3 (T)/PXR3 (T);
XR59 (T) = VXR59(T)/PXR59(T);
XR09 (T) = XR01(T)+XR24(T)+XR3(T)+XR59(T);

TWX01(T) = XF01(T)+XS01(T)+XR01(T);
TWX24(T) = XF24(T)+XS24(T)+XR24(T);
TWX3 (T) = XF3 (T)+XS3 (T)+XR3 (T);
TWX59(T) = XF59(T)+XS59(T)+XR59(T);
TWX09(T) = TWX01(T)+TWX24(T)+TWX3(T)+TWX59(T);

TWPX01(T) = TWVX01(T)/TWX01(T);
TWPX24(T) = TWVX24(T)/TWX24(T);
TWPX3 (T) = TWVX3 (T)/TWX3 (T);
TWPX59(T) = TWVX59(T)/TWX59(T);
TWPX09(T) = TWVX09(T)/TWX09(T);
```

```
/*********************************************************************/
/*                                                                   */
/*                          ROW MODEL                                */
/*                                                                   */
/*********************************************************************/

SOURCE FILE: RTIM_SER;    /* TIME-SERIES (HISTORICAL) DATA FOR ROW     */

RTIM_SER IS FILE (RTS_REC(10));       /* FILE CONTAINING TIME-SERIES DATA */
                                      /* FOR ROW-CVS VARIABLE             */

  RTS_REC IS RECORD(RVNAME,RVNUM,RNUM_PDS,RTS_DATA(1:9));

    RVNAME    IS FIELD (CHAR(8));     /* NAME OF  VARIABLE              */
    RVNUM     IS FIELD (PIC'9999');   /* NUMERIC IDENTIFIER OF VARIABLE */
    RNUM_PDS  IS FIELD (PIC'999');    /* NUMBER OF PDS OF TIME-SERIES   */
                                      /* DATA FOR THIS VARIABLE         */
    RTS_DATA  IS FIELD                /* ARRAY OF TIME SERIES DATA FOR  */
             (PIC'S(5)9.V(3)9');      /*  THIS VAR.; ONE DATUM PER PD.  */
    SIZE.RTS_DATA = RNUM_PDS;         /* THERE IS ONE TIME-SERIES DATUM */
                                      /*  FOR EACH PD. FOR A GIVEN VAR. */

(VMR01,VMR24,VMR3,VMR59,VMR09,PXR01,PXR24,PXR3,PXR59,PXR09)
                  ARE FIELD (PIC'BB(5)-9V.(4)9');

/* THE SOLUTION FOR ROW IS GIVEN EXOGENOUSLY IN THE FOLLOWING EQUATIONS */

VMR01(T) = RTS_DATA(1,T);  VMR24(T) = RTS_DATA(2,T); VMR3(T) = RTS_DATA(3,T);
VMR59(T) = RTS_DATA(4,T);  VMR09(T) = RTS_DATA(5,T);
PXR01(T) = RTS_DATA(6,T);  PXR24(T) = RTS_DATA(7,T); PXR3(T) = RTS_DATA(8,T);
PXR59(T) = RTS_DATA(9,T);  PXR09(T) = RTS_DATA(10,T);
```

```
/*****************************************************************/
/*                                                               */
/*                      FRENCH MODEL                             */
/*                                                               */
/*****************************************************************/

SOURCE FILE: FCOEFF;      /* EQUATION COEFFICIENTS FOR FRANCE          */

FCOEFF IS FILE (FCO_REC(129));       /* FILE OF COEFFS. OF MODEL EQUATIONS    */
  FCO_REC IS RECORD(FA);
   FA IS FIELD (DEC FLOAT(10));

SOURCE FILE: FCONSADJ;    /* CONSTANT ADJUSTMENTS FOR FRANCE           */

FCONSADJ IS FILE (FCA_REC(143));     /* FILE OF CONSTANT ADJUSTMENTS         */
  FCA_REC IS RECORD(FCA(5));        /* ARRAY OF ADJUSTMENTS FOR EACH VAR   */
  FCA IS FIELD (DEC FLOAT(10));

SOURCE FILE: FTIM_SER;    /* TIME-SERIES (HISTORICAL) DATA FOR FRANCE  */

FTIM_SER IS FILE (FTS_REC(142));   /* FILE OF TIME SERIES DATA FOR FRANCE */

  FTS_REC IS RECORD(FVNAME,FVNUM,FNUM_PDS,FTS_DATA(1:9));

    FVNAME    IS FIELD (CHAR(8));     /* NAME OF MODEL VARIABLE         */
    FVNUM     IS FIELD (PIC'9999');   /* NUMERIC IDENTIFIER OF VARIABLE */
    FNUM_PDS  IS FIELD (PIC'999');    /* NUMBER OF PDS OF TIME-SERIES   */
                                      /*  DATA FOR THIS VARIABLE        */
    FTS_DATA  IS FIELD                /* ARRAY OF TIME SERIES DATA FOR  */
            (PIC'S(5)9.V(3)9');       /*  THIS VAR.; ONE DATUM PER PD.  */
    SIZE.FTS_DATA = FNUM_PDS;         /* THERE IS ONE TIME-SERIES DATUM */
                                      /*  FOR EACH PD. FOR A GIVEN VAR. */

TARGET FILE: FS;          /* SIMULATION RESULTS FOR FRANCE             */

FS IS FILE (FSGRP);           /* SOLUTION FILE FOR FRANCE */

  FSGRP IS GROUP(FHDR,T1,FP1,T2,FP2,T3,FP3,T4,FP4,T5,FP5,T6,FP6,T7,FP7,T8,FP8);
  FHDR IS RECORD(TITLE);
  FS.TITLE = 'FRANCE: JOINT LINK/MODEL ECONOMETRIC RESEARCH GROUP';
  T8 IS RECORD(LAB8(2));        LAB8 IS FIELD(CHAR(200));

(FS.LAB1(1),FS.LAB2(1),FS.LAB3(1),FS.LAB4(1),
 FS.LAB5(1),FS.LAB6(1),FS.LAB7(1),FS.LAB8(1)) = ' ';

FS.LAB1(2) = '          ZVM         VM          ZLCH        CH          '||
          'ZWC1         WC1         ZWC2        WC2         ZPVENF      '||
          'PVENF        ZPVAD       PVAD        ZPXTOT';
FS.LAB2(2) = '          PXTOT       ZPCTOT      PCTOT       RDISV       '||
          'ZCMV         CMV         VST         ZLXTOT      XTOT        '||
          'CM           DFCUM       ZLBMTOT     BMTOT';
FS.LAB3(2) = '          PIB         COTOT       ZPVM        PVM         '||
          'XTOTV        BMTOTV      PVTOT       PIBV        ZPOPAOC     '||
          'POPAOC       ZPCM        PCM         ZPOPADI';
FS.LAB4(2) = '          POPADI      DPLAN2      ZVENFV      VENFV       '||
          'VENF         ZPOPASA     POPASA      ZREVNSV     REVNSV      '||
          'REBRUV       RAPDIB      PPIB        POPTOT';
FS.LAB5(2) = '          DURTRA      BALG63      CONTIN      BSAL68      '||
          'B6263        SSALN       PBMTOT      TTDENF      BTC69       '||
          'POPAAG       PRESSOC     WT          PWT';
```

```
FS.LAB6(2) = '          DEVAL4      T491        BSUAL       BTC58       '||
         'CAD          CIF         EFFISC      VAD         VIF         '||
         'DEVAL1       SUSTO       TTIENF      SUSTOV';
FS.LAB7(2) = '          PVST        POPAAD      ZPIBPLA     PIBPLAN     '||
         'NSERVL       FTBL        FCBL        PX01        PX24        '||
         'PX3          PX59        M01         M24';
FS.LAB8(2) = '          M3          M59         DFT         DFTV        '||
         'DFI          PDFT        X01         X24         X3          '||
         'X59          FEXRL       GDPL        PGDPL';

 FP1  IS GROUP (FREC1 (1:9)); FP2  IS GROUP (FREC2 (1:9));
 FP3  IS GROUP (FREC3 (1:9)); FP4  IS GROUP (FREC4 (1:9));
 FP5  IS GROUP (FREC5 (1:9)); FP6  IS GROUP (FREC6 (1:9));
 FP7  IS GROUP (FREC7 (1:9)); FP8  IS GROUP (FREC8 (1:9));

 (SIZE.FREC1,SIZE.FREC2,SIZE.FREC3,SIZE.FREC4,
  SIZE.FREC5,SIZE.FREC6,SIZE.FREC7,SIZE.FREC8) = PD_SIM;

  FREC1 IS RECORD(
YR1,ZVM,VM,ZLCH,CH,ZWC1,WC1,ZWC2,WC2,ZPVENF,PVENF,ZPVAD,PVAD,ZPXTOT);
  FREC2 IS RECORD(
YR2,PXTOT,ZPCTOT,PCTOT,RDISV,ZCMV,CMV,VST,ZLXTOT,XTOT,CM,DFCUM,ZLBMTOT,BMTOT);
  FREC3 IS RECORD(
YR3,PIB,COTOT,ZPVM,PVM,XTOTV,BMTOTV,PVTOT,PIBV,ZPOPAOC,POPAOC,ZPCM,PCM,ZPOPADI);
  FREC4 IS RECORD( YR4,POPADI,DPLAN2,
ZVENFV,VENFV,VENF,ZPOPASA,POPASA,ZREVNSV,REVNSV,REBRUV,RAPDIB,PPIB,POPTOT);
  FREC5 IS RECORD( YR5,DURTRA,
BALG63,CONTIN,BSAL68,B6263,SSSALN,PBMTOT,TTDENF,BTC69,POPAAG,PRESSOC,WT,PWT);
  FREC6 IS RECORD(
YR6,DEVAL4,T491,BSUAL,BTC58,CAD,CIF,EFFISC,VAD,VIF,DEVAL1,SUSTO,TTIENF,SUSTOV);
  FREC7 IS RECORD(
YR7,PVST,POPAAD,ZPIBPLA,PIBPLAN,NSERVL,FTBL,FCBL,PX01,PX24,PX3,PX59,M01,M24);
  FREC8 IS RECORD(
YR8,M3,M59,DFT,DFTV,DFI,PDFT,X01,X24,X3,X59,FEXRL,GDPL,PGDPL);

(FS.YR1(T),FS.YR2(T),FS.YR3(T),FS.YR4(T),
 FS.YR5(T),FS.YR6(T),FS.YR7(T),FS.YR8(T)) = BEG_YR + T - 1;

(ZVM,VM,ZLCH,CH,ZWC1,WC1,ZWC2,WC2,ZPVENF,PVENF,ZPVAD,PVAD,ZPXTOT,PXTOT,
 ZPCTOT,PCTOT,RDISV,ZCMV,CMV,VST,ZLXTOT,XTOT,CM,DFCUM,ZLBMTOT,BMTOT,
 PIB,COTOT,ZPVM,PVM,XTOTV,BMTOTV,PVTOT,PIBV,ZPOPAOC,POPAOC,ZPCM,PCM,
 ZPOPADI,POPADI,DPLAN2,ZVENFV,VENFV,VENF,ZPOPASA,POPASA,ZREVNSV,REVNSV,
 REBRUV,RAPDIB,PPIB,POPTOT,DURTRA,BALG63,CONTIN,BSAL68,B6263,SSSALN,
 PBMTOT,TTDENF,BTC69,POPAAG,PRESSOC,WT,PWT,DEVAL4,T491,BSUAL,BTC58,CAD,
 CIF,EFFISC,VAD,VIF,DEVAL1,SUSTO,TTIENF,SUSTOV,PVST,POPAAD,ZPIBPLA,PIBPLAN,
 NSERVL,FTBL,FCBL,PX01,PX24,PX3,PX59,M01,M24,M3,M59,DFT,DFTV,DFI,PDFT,X01,
 X24,X3,X59,PXF01,PXF24,PXF3,PXF59,PXF09,VMF01,VMF24,VMF3,VMF59,VMF09,
 FEXRL,GDPL,PGDPL)

                  ARE FIELD (PIC'BB(5)-9V.(4)9');

/* HERE ARE A FEW VARIABLES USED FOR INTERIM CALCULATIONS  */

 (VXLF01,VXLF24,VXLF3,VXLF59,VXLF09)  ARE FIELD (PIC'BB(5)-9V.(4)9');


/* FIRST, WE NOTE THE RELATIONSHIPS OF EXOGENOUS SOLUTION VARIABLES TO    */
/*  INPUT TIME-SERIES DATA                                                */

ZLCH    (T) = FTS_DATA(  3,T);   DPLAN2  (T) = FTS_DATA( 41,T);
```

```
POPTOT  (T) = FTS_DATA( 52,T);   DURTRA  (T) = FTS_DATA( 53,T);
BALG63  (T) = FTS_DATA( 54,T);   CONTIN  (T) = FTS_DATA( 55,T);
BSAL68  (T) = FTS_DATA( 56,T);   B6263   (T) = FTS_DATA( 57,T);
SSSALN  (T) = FTS_DATA( 58,T);
TTDENF  (T) = FTS_DATA( 60,T);   BTC69   (T) = FTS_DATA( 61,T);
POPAAG  (T) = FTS_DATA( 62,T);   PRESSOC (T) = FTS_DATA( 63,T);
DEVAL4  (T) = FTS_DATA( 66,T);   T491    (T) = FTS_DATA( 67,T);
BSUAL   (T) = FTS_DATA( 68,T);   BTC58   (T) = FTS_DATA( 69,T);
CAD     (T) = FTS_DATA( 70,T);   CIF     (T) = FTS_DATA( 71,T);
EFFISC  (T) = FTS_DATA( 72,T);   VAD     (T) = FTS_DATA( 73,T);
VIF     (T) = FTS_DATA( 74,T);   DEVAL1  (T) = FTS_DATA( 75,T);
SUSTO   (T) = FTS_DATA( 76,T);   TTIENF  (T) = FTS_DATA( 77,T);
SUSTOV  (T) = FTS_DATA( 78,T);   PVST    (T) = FTS_DATA( 79,T);
POPAAD  (T) = FTS_DATA( 80,T);   ZPIBPLA (T) = FTS_DATA( 81,T);
DFT     (T) = FTS_DATA(102,T);   DFTV    (T) = FTS_DATA(103,T);
DFI     (T) = FTS_DATA(104,T);   PDFT    (T) = FTS_DATA(105,T);
FEXRL   (T) = FTS_DATA(140,T);

/* NEXT, WE GIVE THE RELATIONSHIPS AMONG ENDOGENOUS VARIABLES.  THESE */
/*  EQUATIONS FORM A SET OF SIMULTANEOUS EQUATIONS TO BE SOLVED      */
/*  ITERATIVELY.                                                     */

BLOCK FRANCMOD:  MAX ITER IS 20, RELATIVE ERROR IS 0.00025;

/* THE FIRST EQUATIONS GIVE THE INWARD LINKAGE RELATIONSHIPS FOR FRANCE  */

X01     (T) = IF (T>LAG) THEN VXF01(T)/PXF01(T) ELSE FTS_DATA(106,T);
X24     (T) = IF (T>LAG) THEN VXF24(T)/PXF24(T) ELSE FTS_DATA(107,T);
X3      (T) = IF (T>LAG) THEN VXF3 (T)/PXF3 (T) ELSE FTS_DATA(108,T);
X59     (T) = IF (T>LAG) THEN VXF59(T)/PXF59(T) ELSE FTS_DATA(109,T);
XTOT    (T) = IF (T>LAG) THEN
              (VXF09(T)/PXF09(T))/(1.145*.181) + FCA(114,T)
              ELSE FTS_DATA(22,T);
PBMTOT  (T) = IF (T>LAG) THEN
              1.0493 * PMF09(T) * 1.178/FEXRL(T) * .181 + FCA(124,T)
              ELSE FTS_DATA(59,T);
PWT     (T) = IF (T>LAG) THEN
              (TWPX09(T)-FCA(139,T)) * 100.
              ELSE FTS_DATA(65,T);
WT      (T) = IF (T>LAG) THEN
              (TWVX09(T)/TWPX09(T)) - FCA(134,T)
              ELSE FTS_DATA(64,T);

/* NEXT, THE RELATIONSHIPS INTERNAL TO THE FRENCH ECONOMY ARE GIVEN  */

ZVM     (T) = IF (T>LAG) THEN
              FA(1)*(CH(T-1)/CH(T-2)-1)+FA(2)*(POPTOT(T-1)/POPTOT(T-2)-1)
              +FA(3)*(PCM(T-1)/PVM(T-1)/PCM(T-2)*PVM(T-2)-1)+FA(4)+FCA(1,T)
              ELSE FTS_DATA(  1,T);
VM      (T) = IF (T>LAG) THEN
              VM(T-1)*(1+ZVM(T))+FCA(57,T)
              ELSE FTS_DATA(  2,T);
PIBPLAN(T) = IF (T>LAG) THEN
              PIBPLAN(T-1)*(1+ZPIBPLA(T))+FCA(58,T)
              ELSE FTS_DATA( 82,T);
COTOT   (T) = IF (T>LAG) THEN
              WC1(T)*POPAOC(T)/PIB(T)+FCA(24,T)
              ELSE FTS_DATA( 28,T);
```

```
ZPVM   (T) = IF (T>LAG) THEN
             FA(65)*(PBMTOT(T)/PBMTOT(T-1)-1)+FA(66)*(TTIENF(T)/
             TTIENF(T-1)-1)+FA(67)+FCA(14,T)-FA(64)
             +FA(64)*WC1(T)/WC1(T-1)  /* +FA(74)*ZVM(T)/ZVM(T-1) */
             ELSE FTS_DATA( 29,T);
PVM    (T) = IF (T>LAG) THEN
             PVM(T-1)*(1+ZPVM(T))+FCA(30,T)
             ELSE FTS_DATA( 30,T);
ZPCM   (T) = IF (T>LAG) THEN
             FA(72)*(PBMTOT(T)/PBMTOT(T-1)-1)+FA(73)*(TTIENF(T)
             /TTIENF(T-1)-1)+FCA(16,T)-FA(71)+FA(71)*WC1(T)/WC1(T-1)
             ELSE FTS_DATA( 37,T);
PCM    (T) = IF (T>LAG) THEN
             PCM(T-1)*(1+ZPCM(T))+FCA(38,T)
             ELSE FTS_DATA( 38,T);
ZPCTOT (T) = IF (T>LAG) THEN
             FA(37)+FCA(8,T)+FA(36)*ZPCM(T)
             ELSE FTS_DATA( 15,T);
PCTOT  (T) = IF (T>LAG) THEN
             PCTOT(T-1)*(1+ZPCTOT(T))+FCA(59,T)
             ELSE FTS_DATA( 16,T);
PIB    (T) = IF (T>LAG) THEN
             CAD(T)+CIF(T)+VAD(T)+VIF(T)+SUSTO(T)+CM(T)+VENF(T)
             +VM(T)+VST(T)+XTOT(T)-BMTOT(T)+FCA(27,T)
             ELSE FTS_DATA( 27,T);
ZPOPAOC(T) = IF (T>LAG) THEN
             FA(69)*(B6263(T)-B6263(T-1))+FA(70)+FCA(15,T)-FA(68)
             +FA(68)*PIB(T)/PIB(T-1)
             ELSE FTS_DATA( 35,T);
POPAOC (T) = IF (T>LAG) THEN
             ((POPAOC(T-1)-POPAAD(T-1))*DURTRA(T-1)/DURTRA(T))
             *(1+ZPOPAOC(T))+POPAAD(T)+FCA(22,T)
             ELSE FTS_DATA( 36,T);
ZPOPADI(T) = IF (T>LAG) THEN
             FA(75)*(DURTRA(T)/DURTRA(T-1)-1)+FA(76)*(POPTOT(T)
             /POPTOT(T-1)-1)+FA(78)*(CONTIN(T)/CONTIN(T-1)-1)
             +FA(80)+FCA(17,T)-FA(77)-FA(79)
             +FA(77)*PIB(T)/PIB(T-1)+FA(79)*WC2(T)*DURTRA(T)
             /PCM(T)/WC2(T-1)/DURTRA(T-1)*PCM(T-1)
             ELSE FTS_DATA( 39,T);
POPADI (T) = IF (T>LAG) THEN
             POPADI(T-1)*(1+ZPOPADI(T))+FCA(40,T)
             ELSE FTS_DATA( 40,T);
ZPOPASA(T) = IF (T>LAG) THEN
             FA(86)*(POPAAG(T)/POPAAG(T-1)-1)+FA(87)*(POPAAD(T)
             /POPAAD(T-1)-1)+FA(89)*(WC2(T-1)*DURTRA(T-1)
             /WC2(T-2)/DURTRA(T-2)-1)+FA(90)+FCA(19,T)-FA(88)
             +FA(88)*(POPAOC(T)-POPAAG(T)-POPAAD(T))/(POPAOC(T-1)
             -POPAAG(T-1)-POPAAD(T-1))
             ELSE FTS_DATA( 45,T);
POPASA (T) = IF (T>LAG) THEN
             POPASA(T-1)*(1+ZPOPASA(T))+FCA(46,T)
             ELSE FTS_DATA( 46,T);
CH     (T) = IF (T>LAG) THEN
             CH(T-1) * (10.0 ** ZLCH(T)) + FCA(23,T)
             ELSE FTS_DATA(  4,T);
VST    (T) = IF (T>LAG) THEN
             FA(43)*PCTOT(T-1)/1.09+FA(44)*PVTOT(T-1)/1.087+FA(45)
             *VST(T-1)+FA(46)+FCA(10,T)+FA(42)*CH(T)*.00357+FCA(23,T)
             ELSE FTS_DATA( 20,T);
```

```
ZWC1   (T) = IF (T>LAG) THEN
             FA(11)*(DURTRA(T)/DURTRA(T-1)-1)+FA(13)*(BSAL68(T)
             -BSAL68(T-1))+FA(14)*(B6263(T)-B6263(T-1))
             +FA(15)*(SSSALN(T)/SSSALN(T-1)-1)+FA(16)+FCA(3,T)
             -FA(10)+FA(10)*CH(T)/CH(T-1)+FA(12)*ZPCM(T)
             ELSE FTS_DATA(  5,T);
WC1    (T) = IF (T>LAG) THEN
             WC1(T-1)*(1+ZWC1(T))+FCA(48,T)
             ELSE FTS_DATA(  6,T);
ZWC2   (T) = IF (T>LAG) THEN
             FA(18)*(DURTRA(T)/DURTRA(T-1)-1)+FA(20)*(BSAL68(T)
             -BSAL68(T-1))+FA(21)*(B6263(T)-B6263(T-1))+FA(22)+FCA(4,T)-FA(17)
             +FA(17)*CH(T)/CH(T-1)+FA(19)*ZPCM(T)
             ELSE FTS_DATA(  7,T);
WC2    (T) = IF (T>LAG) THEN
             WC2(T-1)*(1+ZWC2(T))+FCA(60,T)
             ELSE FTS_DATA(  8,T);
ZPVENF (T) = IF (T>LAG) THEN
             FA(24)*(PBMTOT(T)/PBMTOT(T-1)-1)+FA(25)
             *(TTDENF(T-1)/TTDENF(T-2)-1)+FCA(5,T)+FA(23)*ZWC1(T)
             ELSE FTS_DATA(  9,T);
PVENF  (T) = IF (T>LAG) THEN
             PVENF(T-1)*(1+ZPVENF(T))+FCA(61,T)
             ELSE FTS_DATA( 10,T);
ZPVAD  (T) = IF (T>LAG) THEN
             FA(28)*(PBMTOT(T)/PBMTOT(T-1)-1)+FCA(6,T)+FA(27)*ZWC1(T)
             ELSE FTS_DATA( 11,T);
PVAD   (T) = IF (T>LAG) THEN
             PVAD(T-1)*(1+ZPVAD(T))+FCA(49,T)
             ELSE FTS_DATA( 12,T);
ZPXTOT (T) = IF (T>LAG) THEN
             FA(31)*(PBMTOT(T)/PBMTOT(T-1)-1)+FA(32)
             *(TTDENF(T)/TTDENF(T-1)-1)+FA(33)*(XTOTV(T-1)
             /BMTOTV(T-1)/XTOTV(T-2)*BMTOTV(T-2)-1)
             +FA(34)*(BTC69(T)-BTC69(T-1))+FA(35)+FCA(7,T)+FA(30)*ZWC1(T)
             ELSE FTS_DATA( 13,T);
ZLXTOT (T) = IF (T>LAG) THEN
             FA(47)*(WT(T)/WT(T-1)-1)+FA(49)*(DEVAL4(T)
             /DEVAL4(T-1)-1)+FA(50)*(BSUAL(T)-BSUAL(T-1))+FCA(11,T)-FA(48)
             +FA(48)*PXTOT(T)/PWT(T)/PXTOT(T-1)*PWT(T-1)
             ELSE FTS_DATA( 21,T);
PX01   (T) = IF (T>LAG) THEN
             (.659*WC1(T)* 0.001+.902*TWPX01(T))*1.117*.204/FEXRL(T)+FCA(86,T)
             ELSE FTS_DATA(86,T);
PX24    (T) = IF (T>LAG) THEN
              (1.*TWPX24(T))*1.038*.204/FEXRL(T)+FCA(87,T)
              ELSE FTS_DATA(87,T);
PX3    (T) = IF (T>LAG) THEN
             (.6*TWPX3(T))*.8873*.204/FEXRL(T)+FCA(88,T)
             ELSE FTS_DATA(88,T);
PX59   (T) = IF (T>LAG) THEN
             (.1902 + .844*WC1(T)* 0.001 + .6675*TWPX59(T))*1.153*.204/FEXRL(T)
               + FCA(89,T)
             ELSE FTS_DATA(89,T);
PXTOT  (T) = IF (T>LAG) THEN
             (PXF09(T)/VXF09(T)) *
             (PX01(T)*X01(T) + PX24(T)*X24(T) + PX3(T)*X3(T) + PX59(T)*X59(T))
             ELSE FTS_DATA(14,T);
XTOTV  (T) = IF (T>LAG) THEN
             XTOT(T)*PXTOT(T)+FCA(31,T)
             ELSE FTS_DATA( 31,T);
```

```
REBRUV (T) = IF (T>LAG) THEN
             WC2(T)*DURTRA(T)*POPASA(T)* 0.001+PRESSOC(T)
             +REVNSV(T)+FCA(51,T)
             ELSE FTS_DATA( 49,T);
RAPDIB (T) = IF (T>LAG) THEN
             FA(95)*T491(T)+FA(96)+FCA(21,T)-FA(94)
             +FA(94)*REBRUV(T)/REBRUV(T-1)
             ELSE FTS_DATA( 50,T);
RDISV  (T) = IF (T>LAG) THEN
             REBRUV(T)*RAPDIB(T)+FCA(52,T)
             ELSE FTS_DATA( 17,T);
ZCMV   (T) = IF (T>LAG) THEN
             -FA(38)-FA(40)+FCA(9,T)
             +FA(38)*RDISV(T)/RDISV(T-1)+FA(39)*ZPCM(T)+FA(40)
             *PRESSOC(T)/REBRUV(T)/PRESSOC(T-1)*REBRUV(T-1)
             ELSE FTS_DATA( 18,T);
CMV    (T) = IF (T>LAG) THEN
             CMV(T-1)*(1+ZCMV(T))+FCA(54,T)
             ELSE FTS_DATA( 19,T);
CM     (T) = IF (T>LAG) THEN
             CMV(T)/PCM(T)+FCA(54,T)
             ELSE FTS_DATA( 23,T);
BMTOTV (T) = IF (T>LAG) THEN
             BMTOT(T)*PBMTOT(T)+FCA(32,T)
             ELSE FTS_DATA( 32,T);
PIBV   (T) = IF (T>LAG) THEN
             (CM(T)+CAD(T)+CIF(T))*PCTOT(T)+VENF(T)
             *PVENF(T)+(VAD(T)+VIF(T))*PVAD(T)* 0.01
             +VM(T)*PVM(T)+VST(T)*PVST(T)+XTOTV(T)
             -BMTOTV(T)+SUSTOV(T)+FCA(34,T)
             ELSE FTS_DATA( 34,T);
ZVENFV (T) = IF (T>LAG) THEN
             FA(82)*(PVENF(T-1)*EFFISC(T-1)/PVENF(T-2)
             /EFFISC(T-2)-1)+FA(83)*DPLAN2(T-1)+FA(84)
             *VENF(T-1)/PIB(T-1)+FA(85)+FCA(18,T)-FA(81)
             +FA(81)*PIBV(T)/PIBV(T-1)
             ELSE FTS_DATA( 42,T);
VENFV  (T) = IF (T>LAG) THEN
             VENFV(T-1)*(1+ZVENFV(T))+FCA(43,T)
             ELSE FTS_DATA( 43,T);
VENF   (T) = IF (T>LAG) THEN
             VENFV(T)/PVENF(T)+FCA(44,T)
             ELSE FTS_DATA( 44,T);
DFCUM  (T) = IF (T>LAG) THEN
             FA(53)*(CAD(T)+CIF(T))+FA(55)*(VAD(T)+VIF(T))+FCA(12,T)
             +FA(52)*CM(T)+FA(54)*VENF(T)+FA(55)*VM(T)
             +FA(56)*VST(T)+FA(57)*XTOT(T)
             ELSE FTS_DATA( 24,T);
ZLBMTOT(T) = IF (T>LAG) THEN
             FA(60)*(DEVAL1(T)/DEVAL1(T-1)-1)+FA(61)
             *(BSUAL(T)-BSUAL(T-1))+FA(62)*(BSAL68(T)
             -BSAL68(T-1))+FCA(13,T)-FA(59)+FA(59)*DFCUM(T)/DFCUM(T-1)
             ELSE FTS_DATA( 25,T);
BMTOT  (T) = IF (T>LAG) THEN
             BMTOT(T-1)*(1+ZLBMTOT(T))+FCA(26,T)
             ELSE FTS_DATA( 26,T);
ZREVNSV(T) = IF (T>LAG) THEN
             FA(93)-FA(91)+FCA(20,T)+FA(91)*PIBV(T)/PIBV(T-1)+FA(92)*ZPOPASA(T)
             ELSE FTS_DATA( 47,T);
```

```
REVNSV (T) = IF (T>LAG) THEN
             REVNSV(T-1)*(1+ZREVNSV(T))+FCA(55,T)
             ELSE FTS_DATA( 48,T);
PVTOT  (T) = IF (T>LAG) THEN
             (VENF(T)*PVENF(T)+VM(T)*PVM(T)+(VAD(T)+VIF(T))
             *PVAD(T)* 0.01)/(VENF(T)+VM(T)+VAD(T)+VIF(T))+FCA(33,T)
             ELSE FTS_DATA( 33,T);
PPIB   (T) = IF (T>LAG) THEN
             PIBV(T)/PIB(T)+FCA(56,T)
             ELSE FTS_DATA( 51,T);

/* NOW, WE SHOW RELATIONSHIPS FOR THE OUTWARD LINKAGES FOR FRANCE */

PXF01  (T) = IF (T>LAG) THEN
             PX01(T)*.8977*FEXRL(T)*5.525+FCA(115,T)
             ELSE FTS_DATA(115,T);
PXF24  (T) = IF (T>LAG) THEN
             PX24(T)*.8877*FEXRL(T)*5.525+FCA(116,T)
             ELSE FTS_DATA(116,T);
PXF3   (T) = IF (T>LAG) THEN
             PX3(T)*.9116*FEXRL(T)*5.525+FCA(117,T)
             ELSE FTS_DATA(117,T);
PXF59  (T) = IF (T>LAG) THEN
             PX59(T)*.880*FEXRL(T)*5.525+FCA(118,T)
             ELSE FTS_DATA(118,T);
VXLF01  (T) = IF (T>LAG) THEN PXF01(T)*X01(T) ELSE FTS_DATA(110,T);
VXLF24  (T) = IF (T>LAG) THEN PXF24(T)*X24(T) ELSE FTS_DATA(111,T);
VXLF59  (T) = IF (T>LAG) THEN PXF59(T)*X59(T) ELSE FTS_DATA(113,T);
VXLF3   (T) = IF (T>LAG) THEN PXF3 (T)*X3 (T) ELSE FTS_DATA(112,T);
VXLF09  (T) = IF (T>LAG) THEN
             VXLF01(T)+VXLF24(T)+VXLF3(T)+VXLF59(T)
             ELSE FTS_DATA(114,T);
PXF09  (T) = IF (T>LAG) THEN
             VXLF09(T)/(X01(T)+X24(T)+X3(T)+X59(T))
             ELSE FTS_DATA(119,T);
M01    (T) = IF (T>LAG) THEN
             .1360 * BMTOT(T) * 1.178 * .181 + FCA(90,T)
             ELSE FCA(90,T);
M24    (T) = IF (T>LAG) THEN
             .1118 * BMTOT(T) * 1.178 * .181 + FCA(91,T)
             ELSE FCA(91,T);
M3     (T) = IF (T>LAG) THEN
             .1226 * BMTOT(T) * 1.178 * .181 + FCA(92,T)
             ELSE FCA(92,T);
M59    (T) = IF (T>LAG) THEN
             .6296 * BMTOT(T) * 1.178 * .181 + FCA(93,T)
             ELSE FCA(93,T);
VMF01  (T) = IF (T>LAG) THEN PMF01(T)*M01(T) ELSE FTS_DATA(120,T);
VMF24  (T) = IF (T>LAG) THEN PMF24(T)*M24(T) ELSE FTS_DATA(121,T);
VMF3   (T) = IF (T>LAG) THEN PMF3 (T)*M3 (T) ELSE FTS_DATA(122,T);
VMF59  (T) = IF (T>LAG) THEN PMF59(T)*M59(T) ELSE FTS_DATA(123,T);
VMF09  (T) = IF (T>LAG) THEN
             VMF01(T)+VMF24(T)+VMF3(T)+VMF59(T)
             ELSE FTS_DATA(124,T);
NSERVL (T) = IF (T>LAG) THEN FEXRL(T) * SUSTOV(T) ELSE FTS_DATA(83,T);
FTBL   (T) = IF (T>LAG) THEN VXLF09(T) - VMF09(T) ELSE FTS_DATA(84,T);
FCBL   (T) = IF (T>LAG) THEN NSERVL(T) + FTBL(T)  ELSE FTS_DATA(85,T);
GDPL   (T) = IF (T>LAG) THEN PIB(T)*1.3*.181      ELSE FTS_DATA(141,T);
PGDPL  (T) = IF (T>LAG) THEN PPIB(T)*.769*FEXRL(T)*5.525
             ELSE FTS_DATA(142,T);
```



```
   END FRANCMOD;
```

```
TARGET FILE: FSTAT;                  /* SPAIN SOLUTION STATISTICS OUTPUT FILE */
FSTAT    IS FILE (FSTGRP);
FSTGRP   IS GROUP(FTGRP,FRECS(18)); /* TITLE GROUP AND DATA RECORD GROUP     */
FTGRP    IS GROUP(FTLE,FYRS);       /* TITLE GROUP: TITLE AND YEARS HEADER   */
  FTLE   IS RECORD(FTE);
   FTE   IS FIELD(CHAR(100));
   FTE   = 'FRANCE: DYNAMIC SIMULATION STATISTICS';
FYRS     IS RECORD(FSP,FYRHDR(1:9));/* YEARS HEADER: SPACER AND YEAR NAMES   */
 FSP     IS FIELD(CHAR(58));
 FSP     = ' ';
 FYRHDR  IS GROUP(FYR,FDEL);        /* YEAR NAMES INCLUDE %CHG COL. TITLES   */
 FYR     IS FIELD(PIC'(5)B(4)9(5)B');
 FDEL    IS FIELD(CHAR(9));
  SIZE.FYRHDR = PD_SIM - LAG;       /* ONE YRHDR FOR EACH DYNAMIC SIM. YEAR  */
  FYR(SUB1)  = BEG_YR + LAG + SUB1 - 1;
  FDEL(SUB1) = IF(SUB1>1) THEN ' %CHG    ' ELSE ' ';
FRECS    IS RECORD(FPREFIX,FYRDAT(1:9));
 FPREFIX IS FIELD(CHAR(54));
 FYRDAT  IS GROUP(FDATA,FDELTA);    /* ACTUAL DATA AND ANNUAL PERCENT CHANGE */
  FDATA  IS FIELD(PIC'(4)B(5)-9V.999');
  FDELTA IS FIELD(PIC'BBB-ZZZV.Z');
   SIZE.FYRDAT(SUB1) = PD_SIM - LAG;

FPREFIX(1) = 'AGG. DEMAND (NOM. FRANCS):   PRIV CONSUMPTION    CMV  ';
FPREFIX(2) = '                             PRIV INVESTMENT     VENFV';
FPREFIX(3) = '                             GROSS DOM PROD      PIBV ';
FPREFIX(4) = 'AGG. DEMAND (REAL FRANCS):   HSEHOLD CONS        CM   ';
FPREFIX(5) = '                             PUBLIC  CONS        CAD  ';
FPREFIX(6) = '                             HSEHOLD INVSTMT     VM   ';
FPREFIX(7) = '                             INDUSTRY INVSTMT    VENF ';
FPREFIX(8) = '                             PUBLIC INVSTMT      VAD  ';
FPREFIX(9) = '                             INVENTORY CHG       VST  ';
FPREFIX(10)= '                             EXPORT GOODS        XTOT ';
FPREFIX(11)= '                             IMPORT GOODS        BMTOT';
FPREFIX(12)= '                             GROSS DOM PROD      PIB  ';
FPREFIX(13)= 'BAL. PAY.  (CUR. DOLLARS):   FOB TRADE BAL       TB   ';
FPREFIX(14)= 'KEY ECONOMIC INDICATORS  :   GDP DEFLATOR        PPIB ';
FPREFIX(15)= '                             PRIV CONS PRICE     PCM  ';
FPREFIX(16)= '                             PRIV INV DEFL       PVENF';
FPREFIX(17)= '                             EXPORTS DEFL        PXTOT';
FPREFIX(18)= '                             IMPORTS DEFL        PBMTOT';
FPREFIX(19)= '                             WAGE RATE INDEX     WC1  ';

IF(T>LAG) THEN FDATA(1,T-LAG)=CMV (T);  IF(T>LAG) THEN FDATA(2,T-LAG)=VENFV(T);
IF(T>LAG) THEN FDATA(3,T-LAG)=PIBV(T);  IF(T>LAG) THEN FDATA(4,T-LAG)=CM  (T);
IF(T>LAG) THEN FDATA(5,T-LAG)=CAD (T);  IF(T>LAG) THEN FDATA(6,T-LAG)=VM  (T);
IF(T>LAG) THEN FDATA(7,T-LAG)=VENF(T);  IF(T>LAG) THEN FDATA(8,T-LAG)=VAD (T);
IF(T>LAG) THEN FDATA(9,T-LAG)=VST (T);  IF(T>LAG) THEN FDATA(10,T-LAG)=XTOT(T);
IF(T>LAG) THEN FDATA(11,T-LAG)=BMTOT(T);
IF(T>LAG) THEN FDATA(12,T-LAG)=PIB (T);  IF(T>LAG) THEN FDATA(13,T-LAG)=FTBL(T);
IF(T>LAG) THEN FDATA(14,T-LAG)=PPIB(T);  IF(T>LAG) THEN FDATA(15,T-LAG)=PCM (T);
IF(T>LAG) THEN FDATA(16,T-LAG)=PVENF(T);IF(T>LAG) THEN FDATA(17,T-LAG)=PXTOT(T);
IF(T>LAG) THEN FDATA(18,T-LAG)=PBMTOT(T); IF(T>LAG) THEN FDATA(19,T-LAG)=WC1(T);

K IS SUBSCRIPT; L IS SUBSCRIPT;
FDELTA(K,L) = IF (L>1) THEN (FDATA(K,L)/FDATA(K,L-1) - 1.0) * 100.0 ELSE 0.0;
```

```
/*********************************************************************/
/*                                                                   */
/*                        SPANISH MODEL                              */
/*                                                                   */
/*********************************************************************/

SOURCE FILE: SCOEFF;      /* EQUATION COEFFICIENTS FOR SPAIN          */

SCOEFF IS FILE (SCO_REC(73));        /* FILE OF COEFFS. FOR SPANISH MODEL  */
  SCO_REC IS RECORD(SA);
   SA IS FIELD (DEC FLOAT(10));

SOURCE FILE: SCONSADJ;    /* CONSTANT ADJUSTMENTS FOR SPAIN           */

SCONSADJ IS FILE (SCA_REC(118));      /* SPAIN CONSTANT ADJUSTMENT FILE  */
  SCA_REC IS RECORD(SCA(5));         /* ARRAY OF ADJUSTMENTS FOR EACH VAR   */
  SCA IS FIELD (DEC FLOAT(10));

SOURCE FILE: STIM_SER;    /* TIME-SERIES (HISTORICAL) DATA FOR SPAIN   */

STIM_SER IS FILE (STS_REC(118));    /* FILE OF TIME SERIES DATA FOR SPAIN */

  STS_REC IS RECORD(SVNAME,SVNUM,SNUM_PDS,STS_DATA(1:9));

    SVNAME    IS FIELD (CHAR(6));     /* NAME OF MODEL VARIABLE         */
    SVNUM     IS FIELD (PIC'9999');   /* NUMERIC IDENTIFIER OF VARIABLE */
    SNUM_PDS  IS FIELD (PIC'9999');   /* NUMBER OF PDS OF TIME-SERIES   */
                                      /*  DATA FOR THIS VARIABLE        */
    STS_DATA  IS FIELD                /* ARRAY OF TIME SERIES DATA FOR  */
             (PIC'S(5)9.V(3)9');      /*  THIS VAR.; ONE DATUM PER PD.  */
    SIZE.STS_DATA = SNUM_PDS;         /* THERE IS ONE TIME-SERIES DATUM */
                                      /*  FOR EACH PD. FOR A GIVEN VAR. */

TARGET FILE: SS;          /* SIMULATION RESULTS FOR SPAIN             */

SS IS FILE (SSGRP);             /* SPAIN SOLUTION FILE */

  SSGRP IS GROUP (SHDR,T1,SP1,T2,SP2,T3,SP3,T4,SP4,T5,SP5,T6,SP6);

  SHDR IS RECORD(TITLE);
  SS.TITLE = 'SPAIN: JOINT LINK/MODEL ECONOMETRIC RESEARCH GROUP';
  (SS.LAB1(1),SS.LAB2(1),SS.LAB3(1),SS.LAB4(1),SS.LAB5(1),SS.LAB6(1)) = ' ';

SS.LAB1(2) = '          CONS        INV         II          EX          '||
         'IM          GOV         GDP         EMP         PGDP        '||
         'PIM         PEX         WR          URATE';
SS.LAB2(2) = '          T           SSC         SSB         DT          '||
         'IT          COMP        GDPC        NATINC      IMC         '||
         'EXC         CURBAL      EXRATE      IM01C';
SS.LAB3(2) = '          IM24C       IM3C        IM59C       SUB         '||
         'IM09C       IM01        IM24        IM3         IM59        '||
         'PIM01       PIM24       PIM3        PIM59';
SS.LAB4(2) = '          PIM09       IMSERC      IMSER       PIMSER      '||
         'EX01C       EX24C       EX3C        EX59C       EX09C       '||
         'EXSERC      EX01        EX24        EX3';
SS.LAB5(2) = '          EX59        EX09        EXSER       PEX01       '||
         'PEX24       PEX3        PEX59       PEXSER      PEX09       '||
         'IM09        CURACC      NATDIN      WTRAN';
SS.LAB6(2) = '          STBL        XSL         MSL         TRANL       '||
         'SCBL        GNPL        PGNPL       SEXRL';
```

```
 SP1 IS GROUP (SREC1(1:9));  SP2 IS GROUP (SREC2(1:9));
 SP3 IS GROUP (SREC3(1:9));  SP4 IS GROUP (SREC4(1:9));
 SP5 IS GROUP (SREC5(1:9));  SP6 IS GROUP (SREC6(1:9));
(SIZE.SREC1,SIZE.SREC2,SIZE.SREC3,
 SIZE.SREC4,SIZE.SREC5,SIZE.SREC6) = PD_SIM;

SREC1 IS RECORD(YR1,CONS,INV,II,EX,IM,GOV,GDP,EMP,PGDP,PIM,PEX,WR,URATE);
SREC2 IS RECORD(YR2,P,SSC,SSB,DT,IT,COMP,GDPC,
                        NATINC,IMC,EXC,CURBAL,EXRATE,IM01C);
SREC3 IS RECORD(
    YR3,IM24C,IM3C,IM59C,SUB,IM09C,IM01,IM24,IM3,IM59,PIM01,PIM24,PIM3,PIM59);
SREC4 IS RECORD(YR4,
   PIM09,IMSERC,IMSER,PIMSER,EX01C,EX24C,EX3C,EX59C,EX09C,EXSERC,EX01,EX24,EX3);
SREC5 IS RECORD(YR5,
  EX59,EX09,EXSER,PEX01,PEX24,PEX3,PEX59,PEXSER,PEX09,IM09,CURACC,NATDIN,WTRAN);
SREC6 IS RECORD(YR6,STBL,XSL,MSL,TRANL,SCBL,GNPL,PGNPL,SEXRL);

SS.YR1(T) = BEG_YR + T - 1;          SS.YR2(T) = BEG_YR + T -1;
SS.YR3(T) = BEG_YR + T - 1;          SS.YR4(T) = BEG_YR + T -1;
SS.YR5(T) = BEG_YR + T - 1;          SS.YR6(T) = BEG_YR + T -1;

(CONS,INV,II,EX,IM,GOV,GDP,EMP,PGDP,PIM,PEX,WR,URATE,P,SSC,SSB,DT,IT,
 COMP,GDPC,NATINC,IMC,EXC,CURBAL,EXRATE,IM01C,IM24C,IM3C,IM59C,SUB,
 IM09C,IM01,IM24,IM3,IM59,PIM01,PIM24,PIM3,PIM59,PIM09,IMSERC,IMSER,
 PIMSER,EX01C,EX24C,EX3C,EX59C,EX09C,EXSERC,EX01,EX24,EX3,EX59,EX09,
 EXSER,PEX01,PEX24,PEX3,PEX59,PEXSER,PEX09,IM09,CURACC,NATDIN,WTRAN,
 PXS01,PXS24,PXS3,PXS59,PXS09,VMS01,VMS24,VMS59,VMS3,VMS09,GNPL,PGNPL,
 SEXRL,STBL,XSL,MSL,TRANL,SCBL) ARE FIELD (PIC'BB(5)-9V.(4)9');

/* HERE ARE SOME VARIABLES USED FOR INTERIM MODEL CALCULATIONS  */

(PMLS01,PMLS24,PMLS3,PMLS59,PMLS09,VXLS01,VXLS24,VXLS3,VXLS59,VXLS09)
                        ARE FIELD (PIC'BB(5)-9V.(4)9');

/* FIRST, WE NOTE THE RELATIONSHIPS OF EXOGENOUS SOLUTION VARIABLES TO    */
/*  INPUT TIME-SERIES DATA                                                */

GOV   (T) =  STS_DATA( 6,T);    P     (T) =  STS_DATA(14,T);
EXRATE(T) =  STS_DATA(25,T);    SUB   (T) =  STS_DATA(30,T);
PIMSER(T) =  STS_DATA(43,T);    EXSER (T) =  STS_DATA(55,T);
PEX01 (T) =  STS_DATA(56,T);    PEX3  (T) =  STS_DATA(58,T);
WTRAN (T) =  STS_DATA(65,T);

/* NEXT, WE GIVE THE RELATIONSHIPS AMONG ENDOGENOUS VARIABLES.  THESE */
/*  EQUATIONS FORM A SET OF SIMULTANEOUS EQUATIONS TO BE SOLVED       */
/*  ITERATIVELY.                                                      */

BLOCK SPAINMOD:  MAX ITER IS 20, RELATIVE ERROR IS 0.00025;

/* THE FIRST EQUATIONS ARE THE INWARD LINKAGE RELATIONSHIPS          */

SEXRL (T) = IF (T>LAG) THEN 1/EXRATE(T)
            ELSE STS_DATA(113,T);
EX01  (T) = IF (T>LAG) THEN
            (VXS01(T)/PXS01(T) - SCA(81,T))*.69686
            ELSE STS_DATA(50,T);
EX24  (T) = IF (T>LAG) THEN
            (VXS24(T)/PXS24(T) - SCA(82,T))*.69686
            ELSE STS_DATA(51,T);
EX3   (T) = IF (T>LAG) THEN
            (VXS3 (T)/PXS3 (T) - SCA(83,T))*.69686
            ELSE STS_DATA(52,T);
```

```
EX59  (T) = IF (T>LAG) THEN
            (VXS59(T)/PXS59(T) - SCA(84,T))*.69686
            ELSE STS_DATA(53,T);
PIM01 (T) = IF (T>LAG) THEN
            (PMS01(T) - SCA(96,T))/(.01 * SEXRL(T)*69.686)
            ELSE STS_DATA(36,T);
PIM24 (T) = IF (T>LAG) THEN
            (PMS24(T) - SCA(97,T))/(.01 * SEXRL(T)*69.686)
            ELSE STS_DATA(37,T);
PIM3  (T) = IF (T>LAG) THEN
            (PMS3 (T) - SCA(98,T))/(.01 * SEXRL(T)*69.686)
            ELSE STS_DATA(38,T);
PIM59 (T) = IF (T>LAG) THEN
            (PMS59(T) - SCA(99,T))/(.01 * SEXRL(T)*69.686)
            ELSE STS_DATA(39,T);

/* NOW, WE PRESENT THE ENDOGENOUS EQUATIONS CENTRAL TO THE MODEL       */

CONS  (T) = IF (T>LAG) THEN
            SA(1) + SA(2)*NATDIN(T)/PGDP(T) + SA(3)*CONS(T-1) + SCA(1,T)
            ELSE STS_DATA( 1,T);
INV   (T) = IF (T>LAG) THEN
            SA(4) + SA(5)*GDP(T) + SA(6)*GDP(T-1) + SCA(2,T)
            ELSE STS_DATA( 2,T);
II    (T) = IF (T>LAG) THEN
            SA(7)+ SA(8)*(PGDP(T)-PGDP(T-1))/PGDP(T-1) + SA(9)*(GDP(T)-GDP(T-1))
              + SCA(3,T)
            ELSE STS_DATA( 3,T);
GDP   (T) = IF (T>LAG) THEN
            CONS(T) + INV(T) + II(T) + EX(T) + GOV(T) - IM(T)
            ELSE STS_DATA( 7,T);
EMP   (T) = IF (T>LAG) THEN
            SA(14) + EMP(T-1) + SA(15)*(GDP(T)-GDP(T-1)) + SA(16)*INV(T)
                 + SA(17)*(IM(T)-IM(T-1)) + SCA(8,T)
            ELSE STS_DATA( 8,T);
PGDP  (T) = IF (T>LAG) THEN
            SA(18) + SA(19)*COMP(T)/GDP(T) + SA(20)*PIM(T) + SA(21)*PGDP(T-1)
                + SCA(9,T)
            ELSE STS_DATA( 9,T);
WR    (T) = IF (T>LAG) THEN
            WR(T-1) + WR(T-1)*(SA(26)+SA(27)*(PGDP(T)-PGDP(T-1))/PGDP(T-1)
                 + SA(28)/URATE(T-1)) + SCA(12,T)
            ELSE STS_DATA(12,T);
URATE (T) = IF (T>LAG) THEN
          URATE(T-1)+SA(29)+SA(30)*(GDP(T)-(SA(31)+SA(32)*P(T)+SA(33)*P(T)*P(T)))
                       + SA(34)*WR(T)/PGDP(T) + SCA(13,T)
            ELSE STS_DATA(13,T);
SSC   (T) = IF (T>LAG) THEN
            SA(35) + SA(36)*COMP(T) + SCA(15,T)
            ELSE STS_DATA(15,T);
SSB   (T) = IF (T>LAG) THEN
            SA(37) + SA(38)*WR(T) + SA(39)*URATE(T) + SCA(16,T)
            ELSE STS_DATA(16,T);
DT    (T) = IF (T>LAG) THEN
            EXP(SA(40)+SA(42)*(LOG(DT(T-1))-SA(40)
              - SA(41)*LOG(NATINC(T-1)-120.)) + SA(41)*LOG(NATINC(T)-120.))
              + SCA(17,T)
            ELSE STS_DATA(17,T);
IT    (T) = IF (T>LAG) THEN
            EXP(SA(43)+SA(45)*(LOG(IT(T-1))-SA(43)
              - SA(44)*LOG(GDPC(T-1))) + SA(44)*LOG(GDPC(T))) + SCA(18,T)
            ELSE STS_DATA(18,T);
```

```
COMP  (T) = IF (T>LAG) THEN
            0.001*WR(T)*EMP(T)
            ELSE STS_DATA(19,T);
GDPC  (T) = IF (T>LAG) THEN
            GDP(T)*PGDP(T)
            ELSE STS_DATA(20,T);
NATINC(T) = IF (T>LAG) THEN
            GDPC(T) + SUB(T) + SSB(T) - SSC(T) - DT(T) - IT(T)
            ELSE STS_DATA(21,T);
IM01  (T) = IF (T>LAG) THEN
            SA(50) + SA(51)*GDP(T) + SA(52)*IM01(T-1) + SA(53)*PIM01(T)/PGDP(T)
             + SCA(32,T)
            ELSE STS_DATA(32,T);
IM24  (T) = IF (T>LAG) THEN
            SA(54) + SA(55)*GDP(T) + SA(56)*PIM24(T)/PGDP(T) + SCA(33,T)
            ELSE STS_DATA(33,T);
IM3   (T) = IF (T>LAG) THEN
            SA(57) + SA(59)*GDP(T-1) + SA(58)*GDP(T) + SA(60)*PIM3(T)/PGDP(T)
              + SCA(34,T)
            ELSE STS_DATA(34,T);
IM59  (T) = IF (T>LAG) THEN
            SA(61) + SA(62)*GDP(T) + SA(63)*PIM59(T)/PGDP(T) + SCA(35,T)
            ELSE STS_DATA(35,T);
IMSER (T) = IF (T>LAG) THEN
            SA(64) + SA(65)*GDP(T) + SA(66)*PIMSER(T)/PGDP(T) + SCA(42,T)
            ELSE STS_DATA(42,T);
IM01C (T) = IF (T>LAG) THEN IM01(T)*PIM01(T) ELSE STS_DATA(26,T);
IM24C (T) = IF (T>LAG) THEN IM24(T)*PIM24(T) ELSE STS_DATA(27,T);
IM3C  (T) = IF (T>LAG) THEN IM3 (T)*PIM3 (T) ELSE STS_DATA(28,T);
IM59C (T) = IF (T>LAG) THEN IM59(T)*PIM59(T) ELSE STS_DATA(29,T);
IMSERC(T) = IF (T>LAG) THEN
            IMSER(T)*PIMSER(T)
            ELSE STS_DATA(41,T);
IMC   (T) = IF (T>LAG) THEN
            IM01C(T) + IM24C(T) + IM3C(T) + IM59C(T) + IMSERC(T)
            ELSE STS_DATA(22,T);
IM    (T) = IF (T>LAG) THEN
            IM01(T) + IM24(T) + IM3(T) + IM59(T) + IMSER(T)
            ELSE STS_DATA( 5,T);
PIM   (T) = IF (T>LAG) THEN
            IMC(T)/IM(T)
            ELSE STS_DATA(10,T);
IM09C (T) = IF (T>LAG) THEN IM01C(T) + IM24C(T) + IM3C(T) + IM59C(T)
            ELSE STS_DATA(31,T);
IM09  (T) = IF (T>LAG) THEN IM01 (T) + IM24 (T) + IM3 (T) + IM59 (T)
            ELSE STS_DATA(62,T);
PIM09 (T) = IF (T>LAG) THEN
            IM09C(T)/IM09(T)
            ELSE STS_DATA(40,T);
PEX24 (T) = IF (T>LAG) THEN
            SA(67) + SA(68)*PGDP(T) + SCA(57,T)
            ELSE STS_DATA(57,T);
PEX59 (T) = IF (T>LAG) THEN
            SA(69) + SA(71)*PGDP(T-1) + SA(70)*PGDP(T) + SCA(59,T)
            ELSE STS_DATA(59,T);
PEXSER(T) = IF (T>LAG) THEN
            SA(72) + SA(73)*PGDP(T) + SCA(60,T)
            ELSE STS_DATA(60,T);
```

```
EX01C (T) = IF (T>LAG) THEN EX01(T)*PEX01(T) ELSE STS_DATA(44,T);
EX24C (T) = IF (T>LAG) THEN EX24(T)*PEX24(T) ELSE STS_DATA(45,T);
EX3C  (T) = IF (T>LAG) THEN EX3 (T)*PEX3 (T) ELSE STS_DATA(46,T);
EX59C (T) = IF (T>LAG) THEN EX59(T)*PEX59(T) ELSE STS_DATA(47,T);
EX09C (T) = IF (T>LAG) THEN
            EX01C(T) + EX24C(T) + EX3C(T) + EX59C(T)
            ELSE STS_DATA(48,T);
EXSERC(T) = IF (T>LAG) THEN EXSER(T)*PEXSER(T) ELSE STS_DATA(49,T);
EXC   (T) = IF (T>LAG) THEN EX09C(T)+EXSERC(T) ELSE STS_DATA(23,T);
EX09  (T) = IF (T>LAG) THEN
            EX01(T) + EX24(T) + EX3(T) + EX59(T)
            ELSE STS_DATA(54,T);
EX    (T) = IF (T>LAG) THEN EX09(T) + EXSER(T)   ELSE STS_DATA( 4,T);
PEX09 (T) = IF (T>LAG) THEN EX09C(T)/EX09(T)     ELSE STS_DATA(61,T);
PEX   (T) = IF (T>LAG) THEN EXC(T)/EX(T)         ELSE STS_DATA(11,T);
CURBAL(T) = IF (T>LAG) THEN EXC(T) - IMC(T)      ELSE STS_DATA(24,T);
CURACC(T) = IF (T>LAG) THEN CURBAL(T) + WTRAN(T) ELSE STS_DATA(63,T);
NATDIN(T) = IF (T>LAG) THEN NATINC(T) + WTRAN(T) ELSE STS_DATA(64,T);

/* FOLLOWING ARE EQUATIONS FOR THE OUTWARD LINKAGE RELATIONSHIPS          */

PXS01 (T) = IF (T>LAG) THEN
            PEX01(T) * 0.01 * SEXRL(T)*69.686 + SCA(86,T)
            ELSE STS_DATA(86,T);
PXS24 (T) = IF (T>LAG) THEN
            PEX24(T)* 0.01 * SEXRL(T)*69.686 + SCA(87,T)
            ELSE STS_DATA(87,T);
PXS3  (T) = IF (T>LAG) THEN
            PEX3 (T)* 0.01 * SEXRL(T)*69.686 + SCA(88,T)
            ELSE STS_DATA(88,T);
PXS59 (T) = IF (T>LAG) THEN
            PEX59(T)* 0.01 * SEXRL(T)*69.686 + SCA(89,T)
            ELSE STS_DATA(89,T);
VXLS01 (T) = IF (T>LAG) THEN
            PXS01(T) * (1.435 * EX01(T) + SCA(81,T))
            ELSE STS_DATA(81,T);
VXLS24 (T) = IF (T>LAG) THEN
            PXS24(T) * (1.435 * EX24(T) + SCA(82,T))
            ELSE STS_DATA(82,T);
VXLS3  (T) = IF (T>LAG) THEN
            PXS3 (T) * (1.435 * EX3 (T) + SCA(83,T))
            ELSE STS_DATA(83,T);
VXLS59 (T) = IF (T>LAG) THEN
            PXS59(T) * (1.435 * EX59(T) + SCA(84,T))
            ELSE STS_DATA(84,T);
VXLS09 (T) = IF (T>LAG) THEN
            VXLS01(T) + VXLS24(T) + VXLS3(T) + VXLS59(T)
            ELSE STS_DATA(85,T);
PXS09 (T) = IF (T>LAG) THEN
            VXLS09(T)/(VXLS01(T)/PXS01(T) + VXLS24(T)/PXS24(T) +
                       VXLS3 (T)/PXS3 (T) + VXLS59(T)/PXS59(T))
            ELSE STS_DATA(90,T);
PMLS01 (T) = IF (T>LAG) THEN
            PIM01(T)* 0.01 * SEXRL(T)*69.686 + SCA(96,T)
            ELSE STS_DATA(96,T);
PMLS24 (T) = IF (T>LAG) THEN
            PIM24(T)* 0.01 * SEXRL(T)*69.686 + SCA(97,T)
            ELSE STS_DATA(97,T);
PMLS3  (T) = IF (T>LAG) THEN
            PIM3 (T)* 0.01 * SEXRL(T)*69.686 + SCA(98,T)
            ELSE STS_DATA(98,T);
```

```
PMLS59 (T) = IF (T>LAG) THEN
           PIM59(T)* 0.01 * SEXRL(T)*69.686 + SCA(99,T)
           ELSE STS_DATA(99,T);
VMS01 (T) = IF (T>LAG) THEN
           PMLS01(T) * (.9174 * 1.435 * IM01(T) + SCA(91,T))
           ELSE STS_DATA(91,T);
VMS24 (T) = IF (T>LAG) THEN
           PMLS24(T) * (.9174 * 1.435 * IM24(T) + SCA(92,T))
           ELSE STS_DATA(92,T);
VMS3  (T) = IF (T>LAG) THEN
           PMLS3 (T) * (.9174 * 1.435 * IM3 (T) + SCA(93,T))
           ELSE STS_DATA(93,T);
VMS59 (T) = IF (T>LAG) THEN
           PMLS59(T) * (.9174 * 1.435 * IM59(T) + SCA(94,T))
           ELSE STS_DATA(94,T);
VMS09 (T) = IF (T>LAG) THEN
           VMS01(T) + VMS24(T) + VMS3(T) + VMS59(T)
           ELSE STS_DATA(95,T);
PMLS09 (T) = IF (T>LAG) THEN
           VMS09(T)/(VMS01(T)/PMLS01(T) + VMS24(T)/PMLS24(T) +
                    VMS3 (T)/PMLS3 (T) + VMS59(T)/PMLS59(T))
           ELSE STS_DATA(100,T);
STBL  (T) = IF (T>LAG) THEN VXLS09(T) - VMS09(T)       ELSE STS_DATA(114,T);
XSL   (T) = IF (T>LAG) THEN EXSERC(T) * SEXRL(T)       ELSE STS_DATA(115,T);
MSL   (T) = IF (T>LAG) THEN IMSERC(T) * SEXRL(T)       ELSE STS_DATA(116,T);
TRANL (T) = IF (T>LAG) THEN (100.0 + 100.0) * SEXRL(T) ELSE STS_DATA(117,T);
SCBL  (T) = IF (T>LAG) THEN
           STBL(T) + XSL(T) - MSL(T) - TRANL(T)
           ELSE STS_DATA(118,T);
PGNPL (T) = IF (T>LAG) THEN
           PGDP(T)* 0.01 * SEXRL(T)*69.686
           ELSE STS_DATA(112,T);
GNPL  (T) = IF (T>LAG) THEN
           GDPC(T) * SEXRL(T)/PGNPL(T)
           ELSE STS_DATA(111,T);

 END SPAINMOD;

TARGET FILE: SSTAT;                 /* SPAIN SOLUTION STATISTICS OUTPUT FILE */
SSTAT    IS FILE (SSTGRP);
SSTGRP   IS GROUP(STGRP,SRECS(13)); /* TITLE GROUP AND DATA RECORD GROUP     */
STGRP    IS GROUP(STLE,SYRS);       /* TITLE GROUP: TITLE AND YEARS HEADER   */
  STLE   IS RECORD(STE);
   STE   IS FIELD(CHAR(100));
   STE   = 'SPAIN: DYNAMIC SIMULATION STATISTICS';
SYRS     IS RECORD(SSP,SYRHDR(1:9));/* YEARS HEADER: SPACER AND YEAR NAMES   */
 SSP     IS FIELD(CHAR(58));
 SSP     = ' ';
 SYRHDR  IS GROUP(SYR,SDEL);        /* YEAR NAMES INCLUDE %CHG COL. TITLES   */
 SYR     IS FIELD(PIC'(5)B(4)9(5)B');
 SDEL    IS FIELD(CHAR(9));
  SIZE.SYRHDR = PD_SIM - LAG;       /* ONE YRHDR FOR EACH DYNAMIC SIM. YEAR  */
  SYR(SUB1)  = BEG_YR + LAG + SUB1 - 1;
  SDEL(SUB1) = IF(SUB1>1) THEN ' %CHG    ' ELSE ' ';
SRECS    IS RECORD(SPREFIX,SYRDAT(1:9));
 SPREFIX IS FIELD(CHAR(54));
 SYRDAT  IS GROUP(SDATA,SDELTA);    /* ACTUAL DATA AND ANNUAL PERCENT CHANGE */
  SDATA  IS FIELD(PIC'(4)B(5)-9V.999');
  SDELTA IS FIELD(PIC'BBB-ZZZV.Z');
   SIZE.SYRDAT(SUB1) = PD_SIM - LAG;
```

```
SPREFIX(1) = 'AGG. DEMAND (CUR. PESETA):   GROSS NATL PROD     GDPC ';
SPREFIX(2) = 'AGG. DEMAND (1970 PESETA):   PRIV CONSUMPTION    CONS ';
SPREFIX(3) = '                             PUBL CONSUMPTION    GOV  ';
SPREFIX(4) = '                             PUBL INVESTMENT     II   ';
SPREFIX(5) = '                             PRIV INVENTORY      INV  ';
SPREFIX(6) = '                             EXPORT GOODS        EX09 ';
SPREFIX(7) = '                             IMPORT GOODS        IM09 ';
SPREFIX(8) = '                             GROSS DOM PROD      GDP  ';
SPREFIX(9) = 'BAL. PAY.(CUR US DOLLARS):   FOB TRADE BAL       TB   ';
SPREFIX(10)= 'KEY ECONOMIC INDICATORS  :   GDP DEFLATOR        PGDP ';
SPREFIX(11)= '                             EXPORTS DEFLATOR    PEX  ';
SPREFIX(12)= '                             IMPORTS DEFLATOR    PIM  ';
SPREFIX(13)= '                             UNEMPL. RATE (%)    URATE';

IF(T>LAG) THEN SDATA(1 ,T-LAG)=GDPC(T);  IF(T>LAG) THEN SDATA(2 ,T-LAG)=CONS(T);
IF(T>LAG) THEN SDATA(3 ,T-LAG)=GOV (T);  IF(T>LAG) THEN SDATA(4 ,T-LAG)=INV (T);
IF(T>LAG) THEN SDATA(5 ,T-LAG)=II  (T);  IF(T>LAG) THEN SDATA(6 ,T-LAG)=EX09(T);
IF(T>LAG) THEN SDATA(7 ,T-LAG)=IM09(T);  IF(T>LAG) THEN SDATA(8 ,T-LAG)=GDP (T);
IF(T>LAG) THEN SDATA(9 ,T-LAG)=STBL(T);  IF(T>LAG) THEN SDATA(10,T-LAG)=PGDP(T);
IF(T>LAG) THEN SDATA(11,T-LAG)=PEX (T);  IF(T>LAG) THEN SDATA(12,T-LAG)=PIM (T);
IF(T>LAG) THEN SDATA(13,T-LAG)=URATE(T);

I IS SUBSCRIPT;   J IS SUBSCRIPT;
SDELTA(I,J) = IF (J>1) THEN (SDATA(I,J)/SDATA(I,J-1) - 1.0) * 100.0 ELSE 0.0;

 END LINKMOD;
```